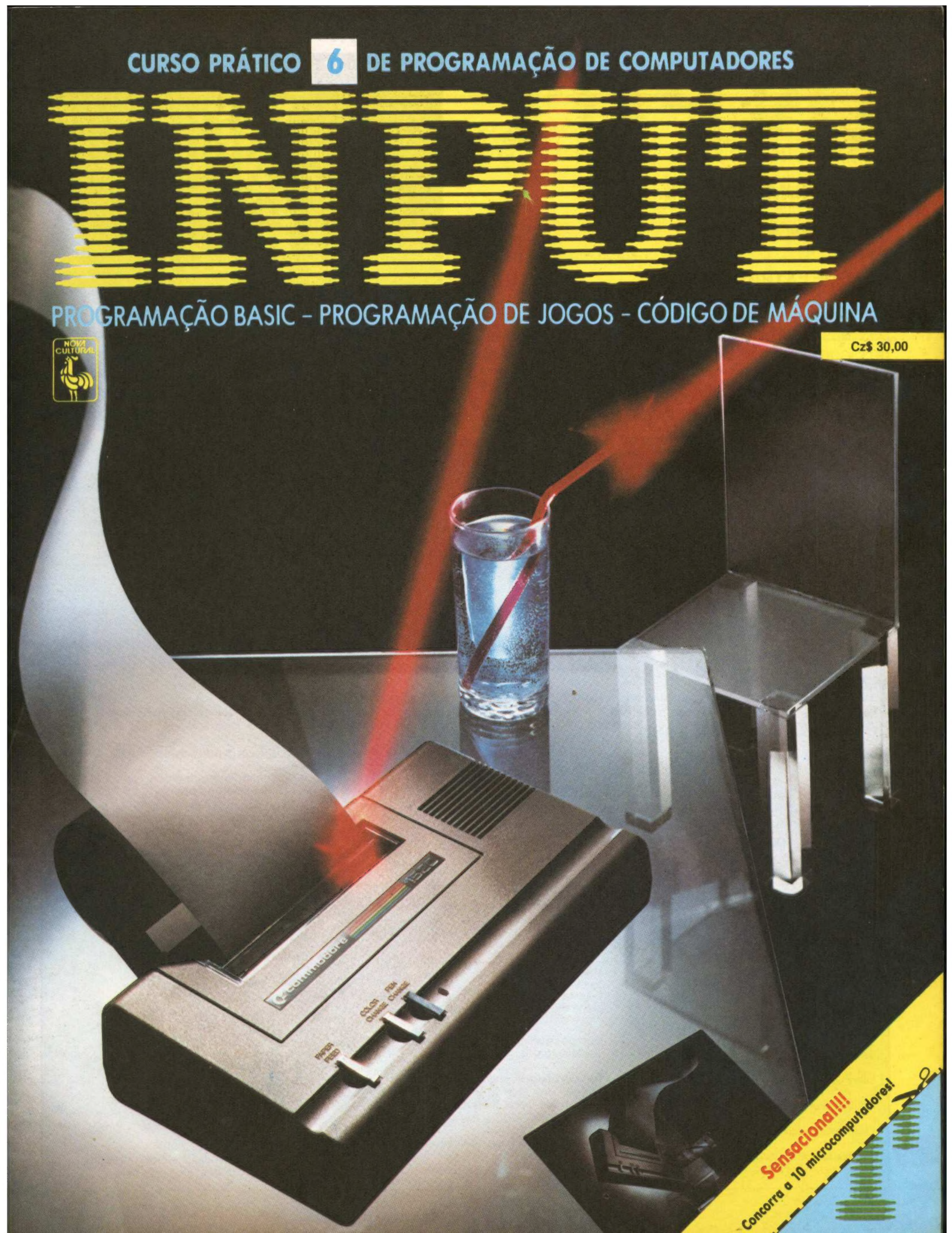
CURSO PRÁTICO 6 DE PROGRAMAÇÃO DE COMPUTADORES

# INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 30,00




Sensacional!!!!
Concorra a 10 microcomputadores!

# INPUT

Vol. 1 N.º 6

## NESTE NÚMERO


**PROGRAMAÇÃO DE JOGOS**

### ATAQUE EXTRATERRESTRE

Tema de muitas obras de ficção científica, as batalhas interplanetárias podem inspirar alguns dos jogos mais fascinantes no computador. Veja como desenhar na tela os elementos do jogo, colocando em ação naves e mísseis extraterrestres..... 101

**CÓDIGO DE MÁQUINA**

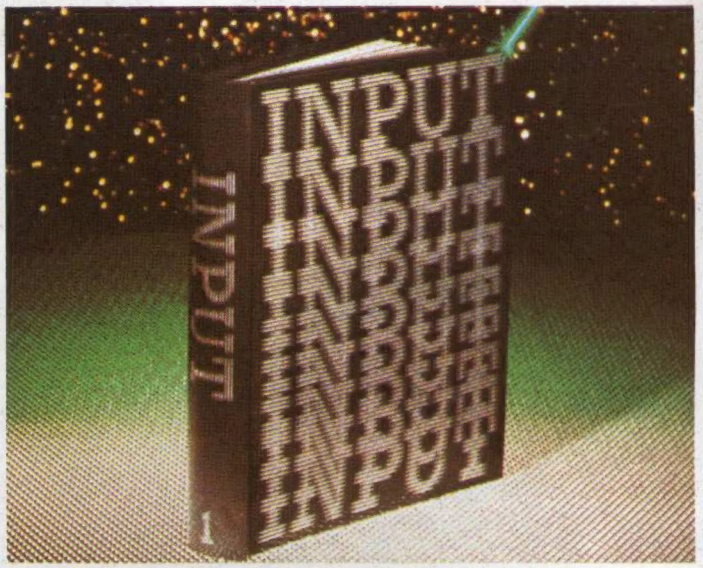
### NO CORAÇÃO DE UM MICRO

O microprocessador é, ao mesmo tempo, o cérebro e o coração de um computador pessoal. Para entender código de máquina, é necessário conhecê-lo. Familiarize-se também com os registros internos e os sinalizadores ..... 109

**PROGRAMAÇÃO BASIC**

### COMO DESENHAR EM BASIC

Os comandos gráficos do BASIC. Gráficos de alta, baixa e média resolução. Como desenhar quadrados, retângulos e círculos. Aplique seus conhecimentos para construir uma casa e um jipe. O comando FLASH..... 113


**PLANO DA OBRA**
"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

**COMPLETE SUA COLEÇÃO**
Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o nº (011) 33 670 DNAP.
Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.
**Atenção:** Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.
**Obs.:** Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

**COLABORE CONOSCO**
Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

**O GRANDE CONCURSO INPUT**

**vai dar 10 microcomputadores HOT BIT BH 8000 da Sharp, através de 2 sorteios pela Loteria Federal. As instruções para você participar desta sensacional promoção estão nos fascículos n.ºs 1 e 7.**


**Editor**
VICTOR CIVITA

**REDAÇÃO**

**Diretor Editorial:** Carmo Chagas

**Editores Executivos:** Antonio José Filho, Berta Sztark Amar

**Editor Chefe:** Paulo de Almeida

**Editor de Texto:** Cláudio A. V. Cavalcanti

**Chefe de Arte:** Carlos Luiz Batista

**Assistentes de Arte:** Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi

**Secretária de Redação/ Coordenadora:** Stefania Crema

**Secretários de Redação:** Marisa Soares de Andrade, Mauro de Queiroz

**COLABORADORES**

**Consultor Editorial Responsável:** Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)

**Execução Editorial:** DATAQUEST Assessoria em Informática Ltda., Campinas, SP

**Tradução, adaptação, programação e redação:** Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira

**Coordenação Geral:** Rejane Felizatti Sabbatini

**Editora de Texto:** Ana Lúcia B. de Lucena

**COMERCIAL**

**Diretor Comercial:** Roberto Martins Silveira

**Gerente Comercial:** Joaquim Celestino da Silva

**Gerente de Circulação:** Denise Maria Mozol



© Marshall Cavendish Limited 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986, 2ª edição, 1987.


Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5988, de 14/12/1973).
Esta obra foi composta pela AM Produções Gráficas Ltda. e impressa pela Companhia Lithographica Ypiranga

# ATAQUE EXTRATERRESTRE

- IDÉIAS PARA UM JOGO ESPACIAL
- COMO DESENHAR NA TELA OS ELEMENTOS DO JOGO
- INCLUA MOVIMENTO
- MÍSSEIS: ELEMENTO DE PERIGO
- COMO CONTROLAR O JOGO

**Batalhas interplanetárias foram sempre um dos pratos fortes dos livros de ficção científica, inspirando até mesmo a estratégia de "Guerra nas Estrelas" proposta por Ronald Reagan. No computador, elas podem ser tema de um jogo fascinante.**

Os jogos para micros ficam muito melhores quando os recursos gráficos de alta resolução do computador são bem aproveitados. Esses gráficos permitem cenas muito mais detalhadas do que os simples caracteres ASCII (ou gráficos de baixa resolução), com os quais aprendemos a lidar até agora. Evidentemente, os programas que utilizam gráficos de alta resolução são bem mais complexos do que aqueles que trabalham apenas com os caracteres pré-programados do teclado, mas os resultados certamente valerão o esforço.

Muitos jogos para computador do tipo videogame envolvem oponentes, invasores extraterrestres ou outros adversários que revidam os disparos ao invés

de esperarem tranqüilamente que você os aniquile.

Agora, vamos ver um jogo chamado *Estação Espacial,* adequado a todos os tipos de computador (com exceção do ZX-81 e do Apple II), que vai ensiná-lo a fazer rotinas gráficas para movimentar aleatoriamente uma nave extraterrestre inimiga na tela, bem como lançar mísseis contra um alvo — no caso, a estação espacial controlada por você.

O jogador dispõe de 4 blindagens que poderá utilizar para se proteger dos ataques dos mísseis inimigos. Entretanto, você não pode mantê-las "ligadas" durante todo o tempo, pois a quantidade de combustível destinada a alimentar sua potência é limitada.

Para tornar o jogo mais difícil, o programa não só movimenta a nave inimiga de um lado para o outro, mas pode também fazê-la desaparecer subitamente dentro do hiperespaço e reaparecer em algum lugar inteiramente inesperado.

Do modo como foi descrito, o jogo não está realmente completo; assim, nada existe nele para medir a passagem do tempo e os pontos, o que pode diminuir o interesse pelo combate. Mas isso pode ser facilmente resolvido utilizando-se os métodos explicados nas páginas 61 a 67.

Embora a nave extraterrestre seja bem parecida com as dos fliperamas, a estação espacial é apenas um esboço. Você pode replanejá-la, se quiser gastar meia hora de modo interessante, utilizando para isso gráficos que você mesmo definirá. Os proprietários de micros da linha MSX podem ainda trabalhar com gráficos programáveis chamados *sprites,* que explicaremos em detalhes mais adiante. No entanto, mantenha o gráfico da sua nova estação espacial dentro da área utilizada por aquela definida neste jogo. Caso contrário, as blindagens defensivas poderão ser superpostas, tornando necessárias modificações mais drásticas do programa.

## T

Digite e execute o programa abaixo:

```
10 PCLEAR 4:PMODE 4,1:PCLS
15 SCREEN 1,1
20 DIM AL(6),BL(6),BO(4)
30 DEFFNZ(X)=SGN(X)*SQR(V*V*X*X
/((127-AX)*(127-AX)+(95-AY)*(95
-AY)))
40 LET PW=250
50 FOR I=0 TO 7:READ A:POKE 153
6+I*32,A:NEXT I
60 GET(0,0)-(7,7),AL,G
65 GOTO 65
250 DATA 24,126,90,126,126,195,
129,129
```

Você verá a nave espacial extraterrestre aparecer na tela.

Os gráficos de alta resolução são apresentados na linha 10. Neste estágio a tela é ligada na linha 15; assim, você poderá ver como o programa funciona, porém, mais tarde, quando o jogo estiver completo, a linha será removida.

Os conjuntos que conterão a nave extraterrestre (AL), o míssil (BO) e um bloco em branco (BL) são dimensionados pela linha 20. A linha 30 utiliza a expressão **DEFFN** que significa *DEFine a FuNction.* Caso seja necessário empregar com freqüência uma longa expressão matemática no programa, a função predefinida o poupará de digitá-la diversas vezes em diferentes pontos do programa. Ela funciona, portanto, como uma espécie de sub-rotina em uma linha só. A expressão matemática na linha 30 é chamada de **FNZ** e será utilizada mais tarde no programa para movimentar um míssil em sentido diagonal, no vídeo. A linha 40 estabelece na tela a posição final de um indicador de combustível.

A linha 50 desenha a nave na tela, lendo os códigos gráficos correspondentes através de instruções **READ** e **DATA** na linha 250, e colocando-os na página de vídeo, no canto superior esquerdo da tela, com comandos **POKE.** A linha 60 usa a função **GET** para capturar esse padrão gráfico definido pelo bloco que vai de 0,7 a 7,7 na tela, e armazená-lo no conjunto denominado AL.

A linha 65 também é temporária. Tudo o que ela faz é manter a tela "congelada", de modo que se possa ver o resultado do processamento feito pelas linhas anteriores. Ela será removida mais tarde, quando o resto do jogo for acrescentado ao programa.

### DESENHE OS MÍSSEIS

Agora, remova a linha 65, digitando o número 65 seguido de <ENTER>: depois, acrescente essas linhas e rode o programa.

```
70 FOR J=0 TO 4:READ A:POKE 153
6+J*32,A:NEXT J
80 GET (0,0)-(4,4),BO,G
85 GOTO 85
260 DATA 32,112,248,112,32
```

O míssil que será lançado pelo E.T. é lido em conjunto de códigos em **DATA,** e desenhado com o auxílio de comandos **POKE** no canto superior esquerdo da tela. Da mesma forma que anteriormente, usamos **GET** para registrar essa forma no conjunto BO.

Não importa que o foguete desenhado na tela se sobreponha ao desenho anterior da nave inimiga, deixando ainda, aparentemente, vestígios dela, pois a linha 80 apenas memoriza com o **GET** a área ocupada pelo míssil e não os seus arredores.

## CONSTRUA A ESTAÇÃO ESPACIAL

Remova a linha 85, digitando 85 seguido de <**ENTER**>, acrescente essas linhas, e depois execute o programa.

```
90 LET AX=RND(248)-1:LET AY=RND
(178)+5
100 PCLS
110 CIRCLE(127,95),12,5:DRAW"BM
127,95;C5S48NUNLNDNR"
115 GOTO 115
```

A linha 90 escolhe uma posição aleatória para a nave E.T. A linha 100 limpa a tela e a linha 110 desenha a estação espacial, que é simplesmente um círculo (comando **CIRCLE**). O comando **DRAW**, no final dessa linha, traça uma cruz bem no meio da estação espacial. Esse comando (**DRAW**) pode ser imaginado como uma sucessão de declarações **LINE**, como já se explicou na lição anterior de programação BASIC. Ele será melhor explicado numa lição posterior.

## DESENHE O INDICADOR DE COMBUSTÍVEL

Remova a linha 115 do mesmo modo que removeu as linhas 65 e 85. Acrescente essas linhas e aparecerão na tela maiores detalhes gráficos.

```
120 DRAW"BM131,87;S4D5BD6BLR3D2
L3D2R3BL12R3U2NL3U2NL3BU6U5G4R3
"
130 DRAW"BM5,1;L4D2NR4D2BE4BR2D
4R3U4BR5L3D2NR3D2R3BE4D4R3"
140 LINE(25,1)-(PW,5),PSET,BF
145 GOTO 145
```

A linha 120 traça na estação espacial um número que corresponde à quantidade de blindagens. Na linha 130 mostramos a palavra FUEL (combustível). Infelizmente, o TRS-Color não consegue escrever caracteres normais numa tela gráfica de alta resolução; por isso, letras e números devem ser literalmente "desenhados" com comandos **DRAW**.

A linha 140 exibe o indicador cheio de combustível para as blindagens, utilizando um método rápido para desenhar retângulos. Usamos a instrução **LINE** para traçar uma linha no alto da tela, indo da posição extrema esquerda (coordenadas 25,1) até a posição do combustível no momento (assinalada pela variável **PW**). O comando **PSET** diz ao TRS-Color para traçar essa linha em branco opaco e, deste modo, estabelece a cor. BF é uma abreviação para *box fill (preencher a caixa)* e "pinta" o retângulo com a cor utilizada pela linha original. Se você quiser traçar um retângulo vazio, utilize B ao invés de BF. Estão agora completos os gráficos de alta resolução para o jogo. O resto do programa está relacionado com o movimento do E.T. e do míssil e a ativação das blindagens.

## MOVIMENTE A NAVE INIMIGA

Existem três sub-rotinas que devem ser acrescentadas ao programa. A primeira diz respeito ao movimento da nave inimiga. Digite-a, mas não a rode ainda porque não acontecerá nada.

```
1000 LET LX=AX:LET LY=AY
1010 IF RND(10)=1 THEN LET AX=R
ND(248)-1:LET AY=RND(178)+5
1020 LET AX=AX+RND(15)-8:LET AY
=AY+RND(15)-8
1030 IF AX>103 AND AX<144 AND A
Y>71 AND AY<112 THEN LET AX=LX:
LET AY=LY
1040 IF AX<0 THEN LET AX=-AX
1050 IF AX>248 THEN LET AX=497-
AX
1060 IF AY<6 THEN LET AY=12-AY
1070 IF AY>184 THEN LET AY=369-
AY
1080 PUT(LX,LY)-(LX+7,LY+7),BL,
PSET
1090 PUT(AX,AY)-(AX+7,AY+7),AL,
PSET
1100 RETURN
```

A nave E.T. (extraterrestre) é controlada de forma semelhante à base de mísseis da lição *Apontar...Fogo!* (páginas 28 a 33). A linha 1000 iguala as coordenadas da última posição da nave às coordenadas da posição atual, antes que ela seja movimentada.

A linha 1010 escolhe um número aleatório para que a nave inimiga possa mudar de posição na tela. Se esse número for 1 (ou seja, em 10% das vezes, em média), então o E.T. saltará para uma nova posição na tela. Se o número não for 1, será escolhida uma nova posição para a nave, distante entre -7 e +7 pontos na direção X (da esquerda para a direita), e entre -7 e +7 pontos na tela, na direção Y (de cima para baixo) — veja a linha 1020. A linha 1030 impede que a nave E.T. sobreponha-se à estação espacial, enquanto as linhas 1040 a 1070 não permitem que ela saia para fora da tela.

A nave invasora é apagada na linha 1080, colocando-se uma série de gráficos em branco nas últimas posições; em seguida, ela é posta na nova posição pela linha 1090. A linha 1100 retorna o programa à linha 160, que, a propósito, ainda não foi inserida.

## COMO DISPARAR OS MÍSSEIS

A próxima sub-rotina serve para decidir, em primeiro lugar, se o míssil deve ser disparado ou não. Caso ocorra o disparo, ela estabelece a posição do míssil na tela e define com que blindagem a defesa vai bloqueá-lo.

```
2000 IF RND(7)<>1 THEN RETURN
2010 V=RND(8)+5:DX=FNZ(127-AX):
DY=FNZ(95-AY)
2020 IF DX<=0 AND DY>=0 THEN LE
T MA=1:GOTO 2050
2030 IF DX<=0 AND DY<=0 THEN LE
T MA=2:GOTO 2050
2040 IF DX>=0 AND DY<=0 THEN LE
T MA=3 ELSE LET MA=4
2050 LET MX=AX:LET MY=AY
2060 PUT(MX,MY)-(MX+4,MY+4),BO,
OR
2070 LET AF=1:RETURN
```

A linha 2000 decide se dispara um míssil. Existem seis chances em uma de que ele seja disparado, mas, se já há um míssil na tela, o programa não pode disparar outro. Se um foguete não puder ser disparado, a sub-rotina terminará.

A linha 2010 estabelece a extensão do deslocamento do míssil na tela — você pode pensar em **V** como se fosse a sua velocidade. O conteúdo de **V** é passado para a função **FNZ** — definida na linha 30 — de modo a calcular a nova posição do míssil na tela.

As linhas 2020 a 2040 averiguam o ponto em que o míssil está na tela e também a blindagem necessária para bloquear o míssil — **MA** é o ângulo de deslocamento do míssil.

A linha 2050 inicia a trajetória do míssil a partir da posição da nave inimiga. E a linha 2060 coloca o foguete na tela antes de a linha 2070 definir a variável sinalizadora **AF = 1**, que diz ao programa se um míssil foi disparado. Uma vez concluídos esses passos, a sub-rotina termina.

A sub-rotina final vai da linha 3000 à 3070. Digite as linhas, mas novamente nada acontecerá se você rodá-las.

```
3000 PUT(MX,MY)-(MX+4,MY+4),BL,
PSET
3010 LET MX=MX+DX:LET MY=MY+DY
3020 IF MX>110 AND MX<140 AND M
Y>79 AND MY<108 THEN GOTO 3050
3030 PUT(MX,MY)-(MX+4,MY+4),BO,
OR
3040 RETURN
3050 IF SH(MA)=0 THEN GOTO 3070
3060 LET AF=0:RETURN
3070 CLS:PRINT @256,"BANG...SUA
 BLINDAGEM ESTAVA DESLIGADA!"
```

O míssil é apagado pela linha 3000, e sua nova posição é computada pela linha 3010. A linha 3020 verifica se o míssil já atingiu a blindagem. Se isso não aconteceu, o programa pula rapidamente para a linha 3050, verificando se a blindagem correta estava posicionada. Quando nenhuma blindagem é encontrada no caminho do foguete, o programa limpa a tela e termina com a mensagem: "BANG! ... SUA BLINDAGEM ESTAVA DESLIGADA".

Se o míssil ainda não atingiu o ponto esperado para alcançar as blindagens da estação espacial, a linha 3050 o colocará rapidamente em nova posição com o comando **PUT**.

A linha 3040 finaliza a rotina. Para terminar, as seguintes linhas completam o nosso programa:

```
150 SCREEN 1,1
160 IF AF=0 THEN GOSUB 2000 ELS
E GOSUB 3000
170 GOSUB 1000
180 FOR J=1 TO 4
190 LET PE=PEEK(338+J):IF PW<25
 THEN LET PE=225
200 IF (255-PE)/16<>SH(J) THEN
LET SH(J)=1-SH(J):CIRCLE(127,95
),16,5*SH(J),1,(J+2)/4,(J+3)/4
210 IF SH(J)=1 THEN LET PW=PW-2.
220 NEXT
230 LINE(PW,1)-(PW+2,5),PRESET,
BF
240 GOTO 160
```

Antes de rodar o programa, remova as linhas 15 e 145. Feito isso, você não verá o que está se desenvolvendo na tela durante a definição dos blocos gráficos. Rodado o programa ocorrerá uma curta pausa, antes que uma exposição completa apareça na tela.

A tela gráfica é agora ligada pela linha 150. A linha 160 checa se um míssil foi disparado. Se **AF = 0**, isso quer dizer que não houve disparos, e o programa salta para a sub-rotina que se refere ao lançamento do míssil, que começa na linha 2000; ou então o programa pula para a sub-rotina que movimenta o míssil; esta última começa na linha 3000. Depois, a nave E.T. movimenta-se. A linha 170 faz com que o programa salte para a sub-rotina na linha 1000.

A parte do programa das linhas 180 a 220 diz respeito à ativação das blindagens. A linha 190 verifica que tecla está sendo pressionada. Se a tecla for um número de 1 a 4, a linha 2000 desenhará a blindagem; se alguma das blindagens for acionada, a linha 210 diminuirá o combustível.

A linha 230 traça um retângulo negro no final do indicador do combustível, dando a impressão de que o suprimento de combustível está acabando, à medida que o valor em PW vai diminuindo. Finalmente, a linha 240 repete tudo de novo.

## S

O programa para o Spectrum utiliza diversos recursos novos que não foram explicados em capítulos anteriores. Por isso, é aconselhável estudar um pouco esses recursos e fazer algumas experiências.

Como de costume, você pode ir checando se os trechos que acabou de entrar funcionam corretamente. O primeiro grupo de linhas define o gráfico da nave extraterrestre e o desenha na tela, quando executado.

```
10 BORDER 0: PAPER 0: INK 6:
BRIGHT 1: CLS
20 FOR n=USR "a" TO USR "b"+7
: READ a: POKE n,a: NEXT n
200 LET ax=INT (RND*32)
210 LET ay=INT (RND*21)+1
220 IF ax>11 AND ax<21 AND ay>
6 AND ay<16 THEN  GOTO 200
490 PRINT  INK 4,AT ay,ax;CHR$
144
800 DATA 60,126,219,219,126,60
,90,153,0,0,24,60,60,24,0,0
```

As linhas 20 e 800 definem a nave e seu míssil (que ainda não é visível). Elas utilizam a técnica de colocar em uma área de memória RAM uma série de blocos gráficos elementares, definidos dentro das declarações **DATA.** Essa área é chamada de *área de gráficos definíveis pelo usuário* (ou UDG, abreviatura em inglês da expressão *user = defined-graphics).* Os comandos **USR "a"** e **USR "b"** definem, para o laço **FOR...NEXT,** os endereços inicial e final dessa área. Os comandos **POKE** fazem esse trabalho.

As linhas 200 e 210 colocam a nave espacial em uma posição aleatória na tela, e a linha 490 a imprime. (O **PRINT CHR$ 144** nessa linha mostra o caractere gráfico associado à tecla A).

A linha 220 pode parecer estranha nesse estágio mas, à medida que o programa progredir, você verá que esta é uma maneira de impedir que a nave inimiga apareça inesperadamente no meio de sua estação espacial, trazendo pânico e destruição.

No momento, você pode sentir vontade de omitir a linha 10, pois pode ser mais difícil ler o programa listado em amarelo sobre uma tela negra. Se quiser fazer isso, lembre-se de reintegrá-la mais tarde, ou a linha 640 (explicada mais adiante) não funcionará.

## CONSTRUA A ESTAÇÃO ESPACIAL

Estas poucas linhas desenharão a estação espacial:

```
110 PRINT AT 10,15;"4 1";AT 12
,15;"3 2"
120 PLOT 132,107: DRAW 25,-25:
DRAW -25,-25: DRAW -25,25:
DRAW  25,25
130 PLOT 107,82: DRAW 50,0:
PLOT 132,57: DRAW 0,50
```

Por enquanto, a estação espacial ainda parece bastante primitiva. Se você quiser projetar e entrar uma mais adequada ou mais bonita, precisará apenas de duas adições ao programa:

- Uma linha extra parecida com a linha 20, mas que comece com **USR "c"** e que continue por tantas letras do alfabeto quanto o tamanho de sua estação espacial necessitar.
- Um conjunto de declarações **DATA** em uma ou mais linhas extras no final do programa, listando os códigos gráficos que compõem a nova estação espacial na tela do computador.

## MOVIMENTE O MÍSSIL

A próxima tarefa é imprimir na tela o míssil da nave E.T. e traçar sua trajetória em direção à estação espacial.

```
150 LET mf=0
300 IF mf=1 THEN  GOTO 400
310 IF RND<.9 THEN  GOTO 420
320 LET mf=1: LET my=ay: LET m
x=ax: LET fy=11-my: LET fx=16-
mx
330 LET b=1: IF ABS fy>ABS fx
THEN  LET b=2
340 IF b=1 THEN  LET sx=SGN fx
: LET sy=SGN fy*ABS (fy/fx)
350 IF b=2 THEN  LET sy=SGN fy
: LET sx=SGN fx*ABS (fx/fy)
400 PRINT AT my,mx;" ": LET my
=my+sy: LET mx=mx+sx: PRINT
INK 5;AT my,mx;CHR$ 145: IF my
>10 AND my<12 AND mx>15 AND mx
<17 THEN  GOTO 700
620 IF RND>.9 THEN  PRINT AT a
y,ax;" ": GOTO 200
630 IF mf=0 THEN  GOTO 300
650 GOTO 300
700 CLS : PRINT  FLASH 1;
PAPER 2;AT 10,1;"BANG!  Voce e
sta sem  protecao!"
```

Toda esta seção do programa, como você pode ver a partir da linha 650, é um laço que o computador percorre diversas vezes sempre que a nave inimiga aparece.

A linha 150 descreve todo o cenário: não existe nenhum míssil vindo em sua direção (até agora...).

A linha 310 decide se a nave lançará um míssil em sua direção, dentro dessa repetição do laço (existe uma chance em nove de que irá disparar).

Se houver um míssil, a linha 320 o colocará na posição inicial (my, mx), no lugar mais óbvio — ou seja, no lugar em que a nave E.T. se encontra no momento (ax, ay). A parte do meio da linha 400 imprime o míssil, usando o caractere gráfico CHR$ 145 (correspondente à tecla B).

O trecho do programa a partir da metade da linha 320 até a linha 400 é um pouco "manhoso". O que ele faz é tomar as coordenadas do ponto médio da estação espacial e as coordenadas referentes à posição atual da nave e subtrair estas das primeiras, de modo a disparar o míssil na direção exata da estação espacial (numa reta passando por dois pontos).

Como alguns dos números envolvidos nesses cálculos podem ser negativos (em movimentos para a esquerda e para baixo) e outros positivos (em movi-

mentos para a direita e para cima), você vai achar difícil entender como o programa funciona, caso não compreenda o que são ABS e SGN, que foram definidos no capítulo anterior. Mas eis aqui algumas dicas para esclarecer o assunto:

A segunda metade da linha 320 deduz a posição atual do míssil (my, mx) do ponto central da estação espacial (posição na tela 11, 16) e chama as coordenadas resultantes (fy e fx).

As linhas 330 e 350, utilizando ABS e SGN, acrescentam um fator de correção de trajetória (sy, sx) para fy e fx. A linha 400 começa imprimindo um espaço em branco sobre a posição anterior do míssil. Em seguida, soma-se sx e sy às coordenadas da posição anterior, de modo a preparar a parte restante da linha 400 para imprimir o míssil em sua nova posição, apenas um passo distante da anterior.

## A NAVE INIMIGA

Agora que a nave inimiga já disparou seu míssil, é hora de fazê-la mover-se; acrescente essas linhas:

```
420 LET xx=ax: LET yy=ay: LET
m=INT (RND*4)
430 IF m=0 AND ax<31 THEN  LET
xx=ax+1
440 IF m=1 AND ax>0 THEN  LET
xx=ax-1
450 IF m=2 AND ay<21 THEN  LET
yy=ay+1
460 IF m=3 AND ay>1 THEN  LET
yy=ay-1
470 IF xx>11 AND xx<21 AND yy>
6 AND yy<16 THEN  GOTO 490
480 PRINT AT ay,ax;" ": LET ax
=xx: LET ay=yy
```

Em primeiro lugar o Spectrum decidirá em que direção a nave E.T. se movimentará. Uma vez que isso seja feito, por um número aleatório na linha 420, os objetivos das linhas 430 a 460 tornam-se óbvios. Estas são as linhas convencionais de animação gráfica. A linha 470 mantém a nave fora da estação. A linha 400 (inserida anteriormente) registra um impacto, direcionando o programa para a linha 700, caso o míssil atinja o meio da estação. Se você não sabe por que essa linha usa **IF my > 10 AND my < 12**, ao invés de um 11 (mais simples) e **IF mx > 15 AND mx < 17**, em vez de 16, basta lembrar-se do seguinte: embora o computador imprima a nave extraterrestre em uma posição com coordenadas inteiras, os números calculados a cada passo do movimento são uma série de *frações decimais*. Assim, torna-se remota a chance de eles tornarem-se 11 e 16.

Finalmente, depois de mais ou menos dez repetições do laço, a linha 620 apaga a nave inimiga de sua posição e começa tudo de novo na linha 200.

## CONSTRUA AS BLINDAGENS

As linhas seguintes constroem as blindagens destinadas a bloquear o avanço do míssil.

```
140 PLOT  INVERSE 1;132,122
500 DIM a(4)
510 LET a$=INKEY$: IF a$=""
THEN  GOTO 600
520 IF a$="1" THEN  LET a(1)=1
530 IF a$="2" THEN  LET a(2)=1
540 IF a$="3" THEN  LET a(3)=1
550 IF a$="4" THEN  LET a(4)=1
560 LET fu=fu-1
600 DRAW  INK a(1)*4, INVERSE
1-a(1),40,-40: DRAW  INK a(2)*
4, INVERSE 1-a(2),-40,-40:
DRAW  INK a(3)*4, INVERSE 1-a(
3),-40,40: DRAW  INK a(4)*4,
INVERSE 1-a(4),40,40
640 IF ATTR (my,mx)=68 THEN
PRINT AT my,mx;" ": LET mf=0
```

À primeira vista também existe alguma coisa estranha com essas linhas. Quatro blindagens, mas apenas uma posição *PLOT* para traçar as linhas?

De fato, o programa utiliza a cor do traço **(INK)**, que é a mesma do fundo **(PAPER)**, para traçar um losango. Somente quando você pressionar uma das teclas numeradas de 1 a 4, uma parte do losango mudará de cor e aparecerá na tela. A linha 640 utiliza o ATTR 68 — o número da cor das blindagens — para repelir o míssil, se ele acertar a blindagem, apagando-o da tela.

## LIMPEZA FINAL

As linhas restantes são muito fáceis de se compreender. Elas estabelecem o suprimento do combustível na linha 100 e o fazem desaparecer até que, no meio da linha 510 (já modificada), as blindagens tornam-se inativas. Lembre-se de reintegrar a linha 10.

```
100 PRINT  PAPER 2; INK 6;AT 0
,0;" COMBUSTIVEL "
160 LET fu=100
510 LET a$=INKEY$: IF a$=""
 THEN  GOTO 600
560 LET fu=fu-1
610 PRINT  PAPER 3; INK 7;AT 0
,13;" ";fu;" "
```

```
10 R=RND(-TIME)
15 SCREEN 2,0
30 DEF FN Z(X)=SGN(X)*SQR(V*V*X.
*X/((127-AX)*(127-AX)+(95-AY)*(
95-AY)))
32 LET PW=250
50 FOR I=0 TO 7
55 READ A:LET A$=A$+CHR$(A)
59 NEXT I
60 SPRITE$(1)=A$
61 PUT SPRITE 1,(8,0)
65 GOTO 65
250 DATA 24,126,90,126,126,195,
129,129
```

Digitando e executando esse programa, você verá a nave espacial extraterrestre aparecer na tela.

A linha 10 serve apenas para introduzir o gerador de números aleatórios, usando o conteúdo do relógio (TIME).

A linha 30 utiliza uma palavra cha-

ve que você ainda não viu: **DEFFN**, que *significa DEFine* a *FuNction*, ou seja, *definição de função*. Se você precisar utilizar com freqüência uma longa expressão matemática no programa, a função predefinida o poupará de digitá-la diversas vezes, em diferentes pontos do programa. Ela funciona, portanto, como uma espécie de sub-rotina em uma linha só. A expressão matemática na linha 30 é chamada de **FNZ** e será utilizada mais tarde no programa para movimentar um míssil em sentido diagonal na tela. A linha 32 estabelece na tela a posição final de um indicador de combustível (variável **PW**).

As linhas 50 a 61 desenham a nave E.T. na tela, lendo os códigos gráficos correspondentes através de instruções

**READ** e **DATA** na linha 250, e colocando-os na página de vídeo, no canto superior esquerdo da tela, com o comando **PUT SPRITE**. A linha 60 usa a função SPRITE para definir este padrão gráfico e armazená-lo no "sprite" número I (por enquanto, não explicaremos detalhadamente o que é

um sprite: veja MICRODICAS na página 108). A linha 61 é temporária, pois apenas mostra o resultado do SPRITE, sendo retirada depois.

A linha 65 também é temporária. Tudo o que ela faz é manter a tela "congelada", de modo que se possa ver o resultado do processamento feito pelas linhas anteriores. Ela será removida mais tarde, quando o resto do jogo for acrescentado ao programa.

## DESENHE OS MÍSSEIS

Agora remova as linhas 61 e 65, digitando os números 65, seguido de <**ENTER**>, e 61. Também seguido de <**ENTER**>. Acrescente depois essas linhas e rode o programa.

```
70 FOR J=0 TO 7
75 READ B:LET BS=BS+CHRS(B)
79 NEXT J
80 SPRITES(0)=BS
81 PUT SPRITE 0,(16,13)
85 GOTO 85
260 DATA 0,0,0,8,28,62,28,8
```

O míssil que será lançado pelo E.T. é lido em um conjunto de códigos em **DATA** e definido como o sprite número 0, nas linhas 70 a 80. O comando **PUT** coloca-o no canto superior esquerdo da tela para você ver, na linha 81. Assim, as linhas 81 e 85 são temporárias.

## A ESTAÇÃO ESPACIAL

Remova as linhas 81 e 85 como anteriormente, e acrescente essas linhas, executando em seguida o programa:

```
90 LET AX=INT(RND(1)*200)-1:LET
 AY=INT(RND(1)*100)+5
110 CIRCLE (127,95),12,15:DRAW
"BM127,95;C15S48NUNLNDNR"
115 GOTO 115
```

A linha 90 escolhe uma posição aleatória para a nave E.T. A linha 110 desenha a estação espacial, que é simplesmente um círculo (comando **CIRCLE**). O comando **DRAW**, no final dessa linha, traça uma cruz bem no meio da estação espacial. Como foi dito anteriormente, o comando **DRAW** pode ser imaginado como uma sucessão de declarações **LINE**. Uma explicação mais detalhada será encontrada em outra lição, mais adiante.

A linha 115 é temporária.

## O INDICADOR DE COMBUSTÍVEL

Remova a linha 115 do mesmo modo que removeu as linhas 61, 65, 81 e 85. Acrescente essas linhas e aparecerão na tela maiores detalhes gráficos.

```
120 DRAW "BM 131,87;S4D5BD6BR3B
D4U5G4R3BDBL12R3U2NL3U2NL3BU10B
L3R3D2L3D2R3"
130 DRAW "BM5,1;L4D2NR4D2BE4BR2
D4R3U4BR5L3D2NR3D2R3BE4D4R3"
140 LINE(30,1)-(PW,5),10,BF
145 GOTO 145
```

A linha 120 traça um número na estação espacial, que corresponde ao número de blindagens. Na linha 130 mostramos a palavra FUEL (combustível). Infelizmente o MSX não consegue escrever caracteres normais numa tela gráfica de alta resolução; por isso letras e números devem ser literalmente "desenhados" com comandos **DRAW**.

A linha 140 exibe o indicador cheio de combustível para as blindagens. Essa linha emprega um método rápido para traçar retângulos. Usamos a instrução **LINE** para traçar uma linha no alto da tela, indo da posição extrema esquerda (coordenadas 30,1) até a posição atual do combustível (assinalada pela variável **PW**). O número 10 diz ao MSX para traçar essa linha em cor branca. BF é uma abreviação para **box fill** e preenche o retângulo com a cor utilizada pela linha original. Se você quer traçar um retângulo vazio, utilize B ao invés de BF.

Estão agora completos os gráficos de alta resolução para o jogo. O resto do programa está relacionado com o movimento do E.T. e do míssil e a ativação das blindagens.

## A NAVE INIMIGA

Existem três sub-rotinas a serem acrescentadas ao programa. A primeira diz respeito ao movimento da nave inimiga. Digite-a, mas não a rode ainda porque nada acontecerá.

```
1000 LET LX=AX:LET LY=AY
1010 IF INT(RND(1)*10)=1 THEN L
ET AX=RND(1)*248-1:LET AY=INT(R
ND(1)*178)+5
1020 LET AX=AX+INT(RND(1)*15)-8
:LET AY=AY+INT(RND(1)*15)-8
1030 IF AX>103 AND AX<144 AND A
Y>71 AND AY<112 THEN LET AX=LX:
LET AY=LY
1040 IF AX<0 THEN LET AX=-AX
1050 IF AX>248 THEN LET AX=497-
AX
1060 IF AY<6 THEN LET AY=12-AY
1070 IF AY>184 THEN LET AY=369-
AY
1080 PUT SPRITE 1,(AX,AY)
1100 RETURN
```

A nave E.T. é controlada de forma semelhante à base de mísseis na lição das páginas 28 a 33. *(Apontar...Fogo!)*. A linha 1000 iguala as coordenadas da última posição da nave às coordenadas da posição atual, antes que ela seja movimentada.

A linha 1010 escolhe um número ao acaso, provocando uma brusca mudança de posição da nave inimiga na tela ("hiperespaço"). Se esse número aleatório for I (ou seja, em 10% das vezes, em média), o E.T. saltará para uma nova posição na tela. Se for diferente de I, será escolhida uma nova posição para a nave, distante entre -7 e +7 pontos na direção X (da esquerda para a direita), e entre -7 e +7 pontos na tela, na direção Y (acima e abaixo) — veja a linha 1020. A linha 1030 impede que a nave E.T. superponha-se ao desenho da estação espacial, enquanto as linhas 1040 a 1070 impedem que ela saia para fora da tela.

A nave invasora é colocada na nova posição pela linha 1080. A linha 1100 faz retornar a sub-rotina.

## DISPARE OS MÍSSEIS

A próxima sub-rotina serve para decidir, em primeiro lugar, se dispara ou não o míssil. Sendo positivo (isto é, se o disparo ocorrer), então ela estabelecerá a posição do míssil na tela e determinará a blindagem que irá bloqueá-lo.

```
2000 IF INT(RND(1)*7)<>1 THEN R
ETURN
2010 LET V=INT(RND(1)*8)+5
2015 LET DX=FNZ(127-AX):LET DY=
FNZ(95-AY)
2020 IF DX<=0 AND DY>=0 THEN LE
T MA=1:GOTO 2050
2030 IF DX<=0 AND DY<=0 THEN LE
T MA=4:GOTO 2050
2040 IF DX>=0 AND DY<=0 THEN LE
T MA=3 ELSE MA=2
2050 LET MX=AX:LET MY=AY
2060 PUT SPRITE 0,(MX,MY)
2070 LET AF=1:RETURN
```

A linha 2000 decide se dispara um míssil. Existem seis chances em uma de que ele seja disparado; mas, se já há um míssil na tela, o programa não pode lançar outro. Se um foguete não puder ser disparado, a sub-rotina terminará.

A linha 2010 estabelece a extensão de deslocamento do míssil na tela — você pode pensar em **V** como se fosse a sua velocidade. O conteúdo de **V** é passado para a função **FNZ** — definida na linha 30 — de modo a calcular a nova posição do míssil na tela.

As linhas 2020 a 2040 investigam o ponto em que o míssil está na tela e também que blindagem será necessária para bloqueá-lo — MA é o ângulo de deslocamento do míssil.

A linha 2050 inicia a trajetória do míssil a partir da posição onde está a nave inimiga. A linha 2060 coloca o míssil na tela antes de a linha 2070 definir a variável sinalizadora **AF = 1**, que diz ao programa que um míssil foi disparado. A sub-rotina termina.

A sub-rotina final vai da linha 3010 à 3080. Digite as linhas, mas observe que novamente não acontecerá nada se você rodá-las nesse estado.

```
3010 LET MX=MX+2*DX:LET MY=MY+2
*DY
3020 IF MX>110 AND MX<140 AND M
Y>79 AND MY<108 THEN GOTO 3050
3030 PUT SPRITE 0,(MX,MY)
3040 RETURN
3050 IF SH(MA)=0 THEN GOTO 3070
3055 PUT SPRITE 0,(-1,-1)
3060 LET AF=0:RETURN
3070 SCREEN 0
3080 LOCATE 0,11:PRINT "BANG!!
- Sua blindagem estava desligad
a"
```

A nova posição do míssil é computada pela linha 3010. A linha 3020 checa se o míssil já atingiu a blindagem. Se isso não acontecer, o programa pula para a linha 3050, com o objetivo de verificar se a blindagem correta estava posicionada. Se não for encontrada nenhuma blindagem no caminho do foguete, o programa limpará a tela e terminará com a mensagem: "BANG! ... SUA BLINDAGEM ESTAVA DESLIGADA".

Se o míssil ainda não atingiu o ponto esperado para as blindagens da estação espacial, então a linha 3050 o coloca em nova posição com o comando **PUT**.

A linha 3040 finaliza a rotina. Finalmente, as seguintes linhas completam o nosso programa:

```
160 IF AF=0 THEN GOSUB 2000 ELS
E GOSUB 3010
170 GOSUB 1000
180 FOR J=1 TO 4
190 K$=INKEY$:IF K$="" THEN K$=
"*"
195 LET PE=ASC(K$)-48
200 IF PE=J THEN SH(J)=1 ELSE S
H(J)=0
201 IF SH(J)<>1 THEN LET SH(J)=
0
205 CIRCLE(127,95),18,15*SH(J),
6.2*(J-1)/4,6.2*(J)/4
210 IFSH(J)=1 THEN LET PW=PW-2
220 NEXT
230 LINE (PW,1)-(PW,5),0,BF
240 GOTO 160
```

Antes de rodar o programa, remova as linhas 115 e 145. Agora que a linha 145 foi removida, você não verá o que está se desenvolvendo na tela durante a definição dos blocos gráficos. Após o programa ter sido rodado, ocorrerá uma curta pausa, antes que uma exposição completa apareça na tela.

A linha 160 verifica se um míssil foi lançado. A equação **AF = 0** indica que nenhum míssil foi disparado, e o programa pula para a sub-rotina que se refere ao lançamento do míssil — que começa na linha 2000 — ou pula para a sub-rotina que movimenta o míssil; esta começa na linha 3010. Depois, a nave E.T. é movimentada. A linha 170 faz com que o programa salte para a sub-rotina que tem início na linha 1000.

A parte do programa das linhas 180 a 220 diz respeito à ativação das blindagens. A linha 190 checa qual tecla está sendo pressionada. Se a tecla for um número de 1 a 4, a linha 2000 desenhará a blindagem; se alguma das blindagens for acionada, a linha 210 diminuirá o combustível.

A linha 230 traça um retângulo negro no final do indicador do combustível, dando a impressão de que o suprimento de combustível está acabando, à medida que o valor em **PW** diminui.

Finalmente, a linha 240 repete tudo novamente, indo para a linha 160.

### MICRO DICAS

**O QUE É UM *SPRITE*?**

Um *sprite* é um tipo de caractere de alta resolução gráfica definido pelo usuário. Ele é utilizado para acrescentar movimento aos gráficos de alta resolução do MSX.

O uso do sprite fornece um aspecto contínuo ao movimento de figuras na tela. Esse movimento pode ser facilmente programado ponto por ponto, como se o objeto que se move fosse apenas um caractere. Além disso, um sprite pode ter até 16x16 pontos de alta resolução, enquanto um caractere comum tem 8x8 pontos. O contorno ou a forma do caractere ou do sprite é definido por pontos "acesos" que assumem a cor de frente, e por pontos "apagados" que assumem a cor de fundo.

É possível definir até 256 sprites pequenos (8x8 pontos) e até 64 sprites grandes (16x16), com padrões diferentes. Todavia, só podem ser mostrados quatro sprites diferentes na mesma linha horizontal de alta resolução. Sprites podem ser agrupados para formar figuras maiores; podem ser sobrepostos para obtermos efeitos tridimensionais; podem, ainda, ter seu tamanho dobrado. Por outro lado, é possível saber quando dois sprites colidiram; tudo isso faz deles caracteres de alta resolução extremamente versáteis.

Programar sprites é realmente mais fácil que utilizar caracteres definidos pelo usuário (compatíveis com o Sinclair Spectrum) ou funções como **PUT**, **GET** (compatíveis com o TRS-Color) e **DRAW** (compatíveis com o Apple II), principalmente quando se deseja uma animação mais sofisticada. Não é necessário, por exemplo, seguir o rastro de um sprite, apagando as posições já ocupadas por ele. Os sprites são compatíveis com outros modos de utilização da tela do MSX: texto em 32 colunas (SCREEN 1) e modo multicor (SCREEN 3). Não se pode, contudo, utilizá-los em modo texto de 40 colunas (SCREEN 0).

# NO CORAÇÃO DE UM MICRO

- O QUE É E O QUE FAZ UMA UCP
- COMUNIQUE-SE COM O MICROPROCESSADOR
- OS REGISTROS INTERNOS
- OS SINALIZADORES
- LOCALIZAÇÕES DE MEMÓRIA

**Para entender a linguagem de máquina é necessário aprender primeiro como funciona o microprocessador, que é, ao mesmo tempo, o coração e o cérebro de um computador pessoal.**

Como você já sabe, os computadores possuem memória. Mas eles não se limitam a lembrar-se das coisas; na verdade, essas máquinas também são capazes de manipular e processar aquilo que memorizam, através da unidade central de processamento (UCP), que é formada por um único circuito integrado *(chip)*, conhecido como *microprocessador*.

Esse circuito integrado é muito mais complexo do que um circuito de memória. As memórias são dispositivos relativamente passivos, que armazenam padrões binários e os desenvolvem. O microprocessador, entretanto, realiza funções bem mais "inteligentes". Ele pode adicionar, subtrair, movimentar informação de uma locação de memória para outra, e deslocar e alternar seus padrões de bits. A programação em linguagem de máquina tem por objetivo "dizer" ao microprocessador (UCP) como ele deve proceder para processar dados e informações, executando uma grande variedade de *operações*.

## O CÉREBRO DA MÁQUINA

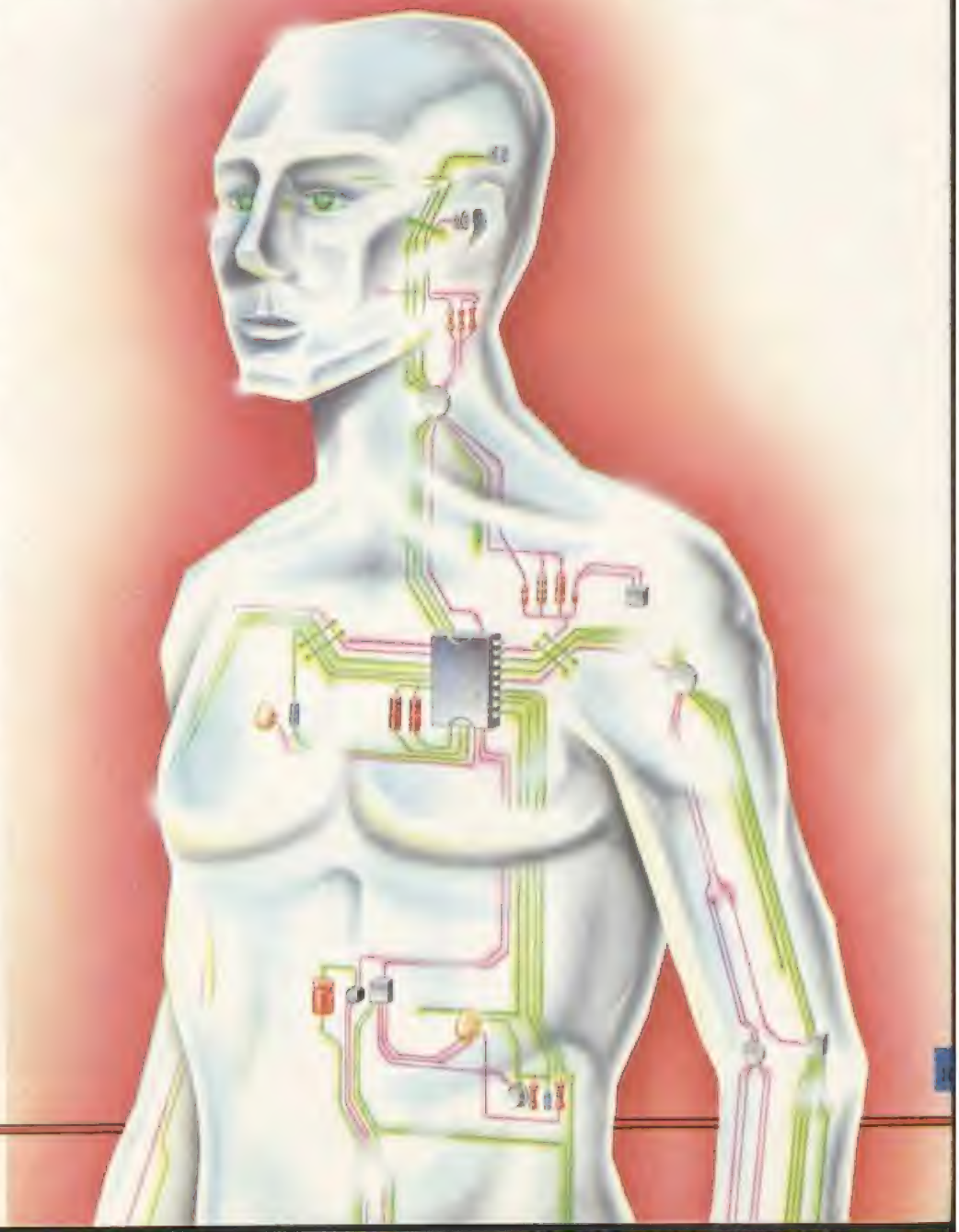
Tudo que acontece no computador está literalmente sob controle direto do microprocessador. Verdadeiro "cérebro", ou centro de controle da máquina, ele efetua todas as operações aritméticas, copia dados de uma locação de memória para outra, atribui funções para áreas de memória e executa os programas. É com o microprocessador que você está se comunicando quando escreve seus programas em código de máquina. Dentro dele existem inúmeras locações especializadas de memória, chamadas *registros*. Cada registro tem uma função específica. E a seleção dos que serão utilizados depende inteiramente das instruções que você fornece quando escreve o seu código de máquina.

Uma das coisas mais importantes sobre os registros é que eles não possuem endereços. Isso faz com que eles se tornem de utilização muito mais rápida.

## OS REGISTROS INTERNOS

As diferentes linhas de computadores pessoais utilizam, geralmente, microprocessadores de marcas diferentes. Assim, os micros das linhas Sinclair, TRS-80 e MSX, por exemplo, empregam o microprocessador de oito bits de comprimento de palavra chamado Zilog Z-80A (este é o seu nome comercial). Já os micros da linha Apple II utilizam um microprocessador também de oito bits mas inteiramente diferente, o 6502.

A complicação para o futuro programador de linguagem de máquina começa aqui: cada um dos diversos chips de microprocessadores citados possui um conjunto diferente de registros internos e de instruções em código de máquina. Em geral, o que se aprende para um ti-



po de chip não se aplica para os outros.

Felizmente, alguns dos registros têm funções similares em todas as máquinas. Os microprocessadores de oito bits possuem um registro chamado PC (Program Counter) de dezesseis bits: é o *contador de programa*. Este armazena o endereço do byte seguinte do código de máquina de um programa a ser executado. Ele "avisa" ao computador onde se encontra, dentro de um programa. E, assim que o byte do programa de código de máquina é executado, o registro PC é aumentado de um.

Todos os microcomputadores de oito bits também têm, pelo menos, um outro registro, chamado *acumulador* de oito bits ou registro A. É o registro utilizado pela UCP para efetuar operações aritméticas. Se você quiser, por exemplo, acrescentar um número a outro (ou subtraí-lo dele), uma das duas parcelas deverá estar no acumulador. Isso também afeta a multiplicação e a divisão — que são apenas combinações de operações lógicas como as adições e as subtrações.

Cada uma dessas máquinas possui também um par de *registros de indexação*. Nos microprocessadores Z-80A estes são registros de dezesseis bits que contêm endereços. Tais endereços podem ser aumentados, diminuídos ou modificados aritmeticamente no acumulador, de modo a fornecer outros endereços. Os micros da linha Apple (microprocessador 6502) possuem registros de indexação de oito bits cujos conteúdos são incorporados a um endereço para fornecerem um endereço adicional.

Conhecido como *endereçamento indexado*, esse tipo de operação é mais comumente utilizado para ler dados de uma tabela, byte por byte. O programa começa com o endereço do primeiro byte da tabela e, a seguir, percorre seu caminho aumentando o registro de indexação de um em um.

## LEVANTE A BANDEIRA

O sinalizador (registro-bandeira, ou *flag-register*, em inglês, numa analogia à bandeira dos taxímetros) é outro registro existente em todos os microprocessadores de 8 bits. Um registro sinalizador tem normalmente oito bits, que podem estar "ligados" (bit 1), ou "desligados" (bit 0), dependendo do resultado de certas operações. Cada bit de um registro sinalizador é chamado de *bandeira*, ou sinalizador.

Todo sinalizador tem um significado próprio. Por exemplo, o bit número 0, ou seja, o bit menos significativo do registro sinalizador (o último bit à direita na convenção normal de escrita), é o bit de transporte ou sinalizador C (*carry*, em inglês). Se a UCP efetuar uma adição, uma subtração, ou qualquer outra operação lógica ou aritmética que exija um bit para ser emprestado ou transportado (o "vai-um" das operações aritméticas), então o sinalizador C será estabelecido em 1.

Os outros bits indicam se o resultado de uma operação foi zero *(o sinalizador de zero)* ou negativo *(o sinalizador de sinal)*, ou ainda se um registro foi excedido em capacidade porque o número era muito grande *(o sinalizador de excesso*, ou *overflow)*. Esses sinalizadores são comuns a todas as máquinas, embora não estejam necessariamente na mesma posição no registro sinalizador. Existem também sinalizadores específicos para cada máquina.

Os sinalizadores são utilizados em instruções de desvio condicional em linguagem de máquina — o equivalente em BASIC ao **IF...THEN**. No programa em código de máquina, podemos escrever instruções que farão com que a máquina salte ou desvie se um sinalizador estiver em 0 ou em 1, por exemplo.

## A PILHA

Comum a todos os computadores, o *indicador da pilha* (em inglês: *stack-pointer)* é um registro destinado a reter o endereço de uma locação de memória RAM. Esse endereço marca o fim de uma área de memória, como todos os indicadores fazem; mas, aqui, essa é uma área especializada de memória, de uso exclusivo da UCP e conhecida como *pilha*. Ela é uma área de rápido acesso, onde a informação pode ser temporariamente armazenada e chamada de volta sem precisar dos procedimentos normais de endereçamento.

A pilha da UCP funciona de modo parecido a uma pilha comum de papéis: toda informação nova que se quer acrescentar a ela deve ser colocada em seu topo. Inversamente, quando se quer tirar dela alguma coisa, é necessário começar pelo topo, retirando as folhas mais altas (não vale tirar um papel do meio ou da parte mais baixa da pilha). A regra é: o último a entrar é o primeiro a sair (em inglês: *last in, first out*). Por isso, esse tipo de pilha é chamado de LIFO. Existem outras espécies como a pilha FIFO *(first in, first out)*. A fila de um supermercado é um exemplo de pilha FIFO...

Uma das funções do microprocessador consiste em colocar dados, endereços ou códigos no topo da pilha da memória. Se ele for instruído para retirar algo da pilha, começará sempre pelos itens mais altos. Isso significa que você não precisa especificar a locação de memória que lhe interessa, ou sua posição na pilha. Existe apenas uma locação do topo de cada vez; assim, o microprocessador não se confundirá.

Mas você, o programador, pode ficar extremamente confuso, se não prestar atenção! É importante lembrar-se de tudo o que existe em cada locação de memória da pilha, quando se estiver fazendo um programa que a utiliza, e certificar-se de que o dado desejado está no topo da pilha.

Agora que você já domina o conceito de pilha LIFO, podemos revelar-lhe um segredo: em todas as máquinas citadas aqui, a base da pilha é dirigida para o topo da RAM, com o resto da pilha acumulado abaixo, item por item. Assim, na verdade, você introduz e tira coisas da parte inferior da pilha!

Isto quer dizer que, quando um novo item é introduzido, o indicador de pilha é diminuído, e, quando um item é tirado, o indicador é aumentado.

Apesar disso, o conceito de pilha se mantém verdadeiro, com a base da pilha fixada por uma variável de sistema, e os itens dos dados adicionados ou subtraídos a partir de um final livre, um de cada vez.

## SUB-ROTINAS

Uma das funções mais comuns de uma pilha é manter a UCP a par de seu lugar em um programa quando uma sub-rotina for executada. Isso pode parecer simples, mas devemos nos lembrar de que uma sub-rotina pode chamar outra; esta, uma outra, e assim por diante. Quando o computador atinge uma instrução que lhe diz para saltar para uma sub-rotina, o conteúdo do contador de programa nesse momento (isto é, onde o programa se encontra no instante em que a instrução de desvio ocorre) é copiado no topo da pilha.

O endereço inicial da sub-rotina é então escrito no registro contador de progama. À medida que cada instrução é executada em uma sub-rotina, o contador de programa é incrementado da maneira normal, até que atinja uma instrução de retorno. Então, o endereço que está no topo da pilha é retirado e colocado de volta no registro contador de programa. Esse processo permite que a UCP reinicie a execução do programa exatamente no ponto onde estava antes de chamar a sub-rotina e passe a incrementar o contador de programa nova-

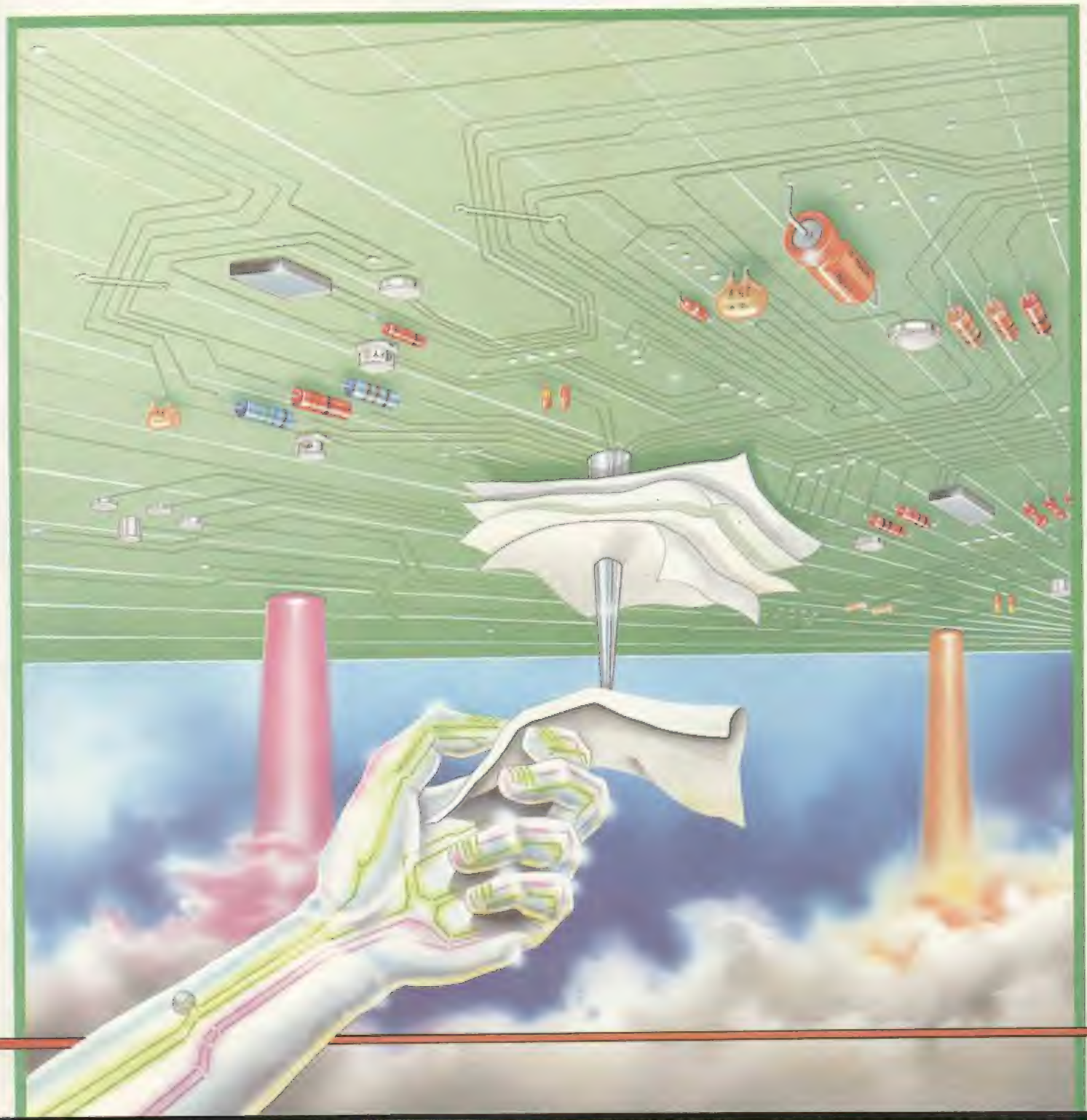
mente. Isso explica por que, em programas BASIC, cada sub-rotina encaixada dentro de uma outra deve estar contida completamente naquela que a precede; do contrário, os endereços incorretos de partida seriam colocados no topo da pilha. O mesmo acontece com os laços de repetição, tipo **FOR...NEXT**, cujos endereços de retorno no programa são armazenados na pilha. Agora que você domina os conceitos básicos da programação de códigos de máquina, descubra como eles se relacionam ao microprocessador específico de seu computador.

## TS5X

O microprocessador dessas máquinas é o Zilog Z80A, de oito bits. Ele tem características exclusivas, como o fato de contar com um grande número de registros internos (21, para ser exato). Estes são também chamados de registros do usuário, porque podem ser controlados diretamente pelo usuário ou pelo programador da máquina.

Existem dois acumuladores de oito bits: A e A'. Esses acumuladores não podem ser utilizados ao mesmo tempo, mas você pode comutá-los para desempenharem simultaneamente duas operações. Contudo, não é aconselhável tentar isso no ZX81. Se a máquina estiver na modalidade *SLOW* (lenta) você poderá perder o conteúdo da tela.

O registro sinalizador F também possui oito bits. O bit na posição 0 é o sinalizador de transporte C; o bit na posição 1 é o sinalizador de subtração N, o qual entra em ação apenas quando operações codificadas em BCD (*decimal codificado em binário* — uma representação especial de números decimais em binário) são utilizadas; o bit 2 é o sina-



lizador de paridade/excedente P/V; o bit 4 é o sinalizador de meio transporte H, que também é utilizado em BCD; o bit 6 é o sinalizador de 0 Z, e o bit 7 é o sinalizador de sinal S. Existe um registro sinalizador alternativo F' que é comutado junto com A'.

Existem seis registros de propósitos gerais — B, C, D, E, H e L — que tanto podem ser utilizados separadamente, como registros de oito bits, quanto em pares, como registros de dezesseis bits (neste caso, são chamados coletivamente de registros BC, DE e HL). Também existe um conjunto alternativo: B 'C', D 'E' e H 'L'. O HL e os registros alternativos H 'L' podem igualmente ser utilizados como acumuladores de dezesseis bits. Contudo, os registros H 'L' devem ter seus valores anteriores restaurados antes de retornarem para BASIC.

O indicador de pilha do Z-80 tem dezesseis bits, assim como os registros de indexação IX e IY. Estes últimos retêm endereços que podem ser aumentados ou deslocados por meio da adição ou subtração de números conhecidos como "deslocamentos" *(off-sets)*, destinados a fornecer um outro endereço. O IY aponta para o centro das variáveis de sistemas e é utilizado pelas rotinas ROM para indexá-las. Se o registro IY for acionado, seu valor original deverá ser restaurado mais tarde ou o computador não funcionará.

Existem mais dois registros de propósitos especiais no microprocessador Z-80: o I e o R. O registro I armazena parte do endereço utilizado para iniciar "rotinas interrompidas". São rotinas que interrompem o curso normal do seu programa de código de máquina a cada 1/50 de segundo, para desempenhar alguma função vital tal como a varredura do teclado.

O registro R renova a memória dinâmica *(refresh)*, mas você não poderá utilizá-lo no Spectrum ou no ZX-81.

A parte inferior da pilha está no RAMTOP (locação máxima da memória RAM em um microcomputador) e pode ocupar uma área tão grande da RAM quanto seja necessário. O indicador de pilha, ou registro SP, possui dezesseis bits; assim, ele armazena todo o endereço do último byte ocupado da pilha.

Os micros da linha Apple II, bem como o TK-2000, utilizam o microprocessador 6502.

Esse chip possui um único acumulador de oito bits e dois registros de indexação de oito bits cada: X e Y. Estes contêm deslocamentos *(off-sets)*, que são acrescentados a um endereço-base para se obter outro endereço. Esse método é muito utilizado para ler uma tabela de dados, por exemplo. O endereço-base, no caso, é o início da área de memória contendo uma tabela. Para ler a tabela, usamos um laço, que incrementa apenas o deslocamento a cada passagem. Com isso, o programa é direcionado para o próximo byte de tabela, cada vez que o laço é repetido.

É possível direcionar um programa em código de máquina para um endereço que contenha um outro endereço — desde que este se encontre na página zero da RAM, pois os endereços indiretos podem ter só oito bits de comprimento (isto limita em 255 o número de memórias endereçáveis por este método). Você pode deslocar um desses dois endereços, mas o registro X deve ser utilizado para deslocar a partir do primeiro endereço, e o registro Y deve deslocar a partir do segundo endereço.

O *registro processador* P de oito bits, também conhecido como *registro de estado (status-register)*, contém um sinalizador. Os sinalizadores de transporte C são os mais importantes e ocupam o bit 0; o sinalizador de 0 Z ocupa o bit 1; o sinalizador de excesso V ocupa o bit 6; e o sinalizador de sinal N ocupa o bit 7.

Existem mais três sinalizadores que você pode utilizar. O sinalizador de decimal D, que é o bit 3, é estabelecido quando o microprocessador efetua operações aritméticas em binário codificado em decimal (BCD). O sinalizador de pausa B e o sinalizador de interrupção I são utilizados em "rotinas de interrupção". Estas interrompem, a intervalos regulares, o fluxo normal dos seus programas em código de máquina para desempenharem funções vitais, como a varredura do teclado.

O 6502 também tem uma pilha que está limitada à página 1, com sua base em $01FF, pois o *indicador de pilha*, ou registro SP, tem apenas 8 bits. Esse indicador pode conter qualquer número de $00 a $FF e fornece o byte menos significativo do primeiro endereço livre localizado abaixo da pilha. O microprocessador já sabe que o bit mais significativo do endereço é 01, pois a pilha está na página 1.

O microprocessador existente nos computadores da linha TRS-Color é do tipo Motorola 6809E de oito bits, operando a uma freqüência de 0,895 MHz. Ele tem dois *acumuladores:* os registros A e B, que podem ser utilizados independentemente, como registros separados de oito bits, ou juntos, como um acumulador D de dezesseis bits.

Ele conta ainda com dois *registros de indexação* de dezesseis bits — X e Y — que retêm os endereços que podem ser aumentados ou deslocados, por meio da adição ou subtração de números (conhecidos como "deslocamentos" ou *off-sets)*, para fornecerem um outro endereço. Freqüentemente, essa forma de programação é utilizada para ler uma tabela de dados a partir de um endereço-base.

O *registro de código de condição* retém os diversos bits sinalizadores. O bit 0 é o sinalizador de transporte C; o bit 1 é o sinalizador de excesso V; o bit 2 é o sinalizador de zero Z; o bit 3 é o sinalizador de sinal N; o bit 5 é o sinalizador de meio transporte, que é ativado apenas com operações aritméticas com BCD (decimal codificado em binário, que é uma representação especial de números decimais); o bit 4 é o sinalizador da máscara de interrupção normal I; o bit 6 é o sinalizador da máscara de interrupção rápida F; e o bit 7 é o sinalizador de total E — todos são utilizados em rotinas de interrupção, que interrompem, a intervalos regulares, o fluxo normal do programa de código de máquina para desempenharem funções decisivas, como atualizar o cronômetro interno, fazer a varredura do teclado para ver se alguma tecla foi apertada etc.

O chip 6809 possui ainda um *registro de página direta* — ou DP — de oito bits. A função desse registro é armazenar o byte mais significativo dos endereços que você escolheu para a página direta, de modo que a locação nessa página possa ser endereçada apenas pelo seu byte menos significativo.

São dois os *indicadores de pilha*, de dezesseis bits cada: S e U. O primeiro revela o último endereço completo no hardware da pilha cuja base está em RAMTOP ou &H7FFF (a menos que tenha sido empurrado para baixo, de modo a deixar espaço a um programa em código de máquina). O indicador U, por sua vez, é utilizado com a pilha do usuário, cuja base deve ser normalmente definida pelo programador. A pilha do usuário é pouco utilizada; assim, o registro U serve como um registro extra de indexação de dezesseis bits pelos códigos de máquinas mais sofisticados no TRS-Color ou pelo programador de linguagem Assembly.

# COMO DESENHAR EM BASIC

- UTILIZE OS COMANDOS GRÁFICOS DO BASIC
- DESENHE QUADRADOS, TRIÂNGULOS E CÍRCULOS
- USE **FLASH** PARA ANIMAR PROGRAMAS NO SPECTRUM
- GRÁFICOS NO APPLE E NO MSX

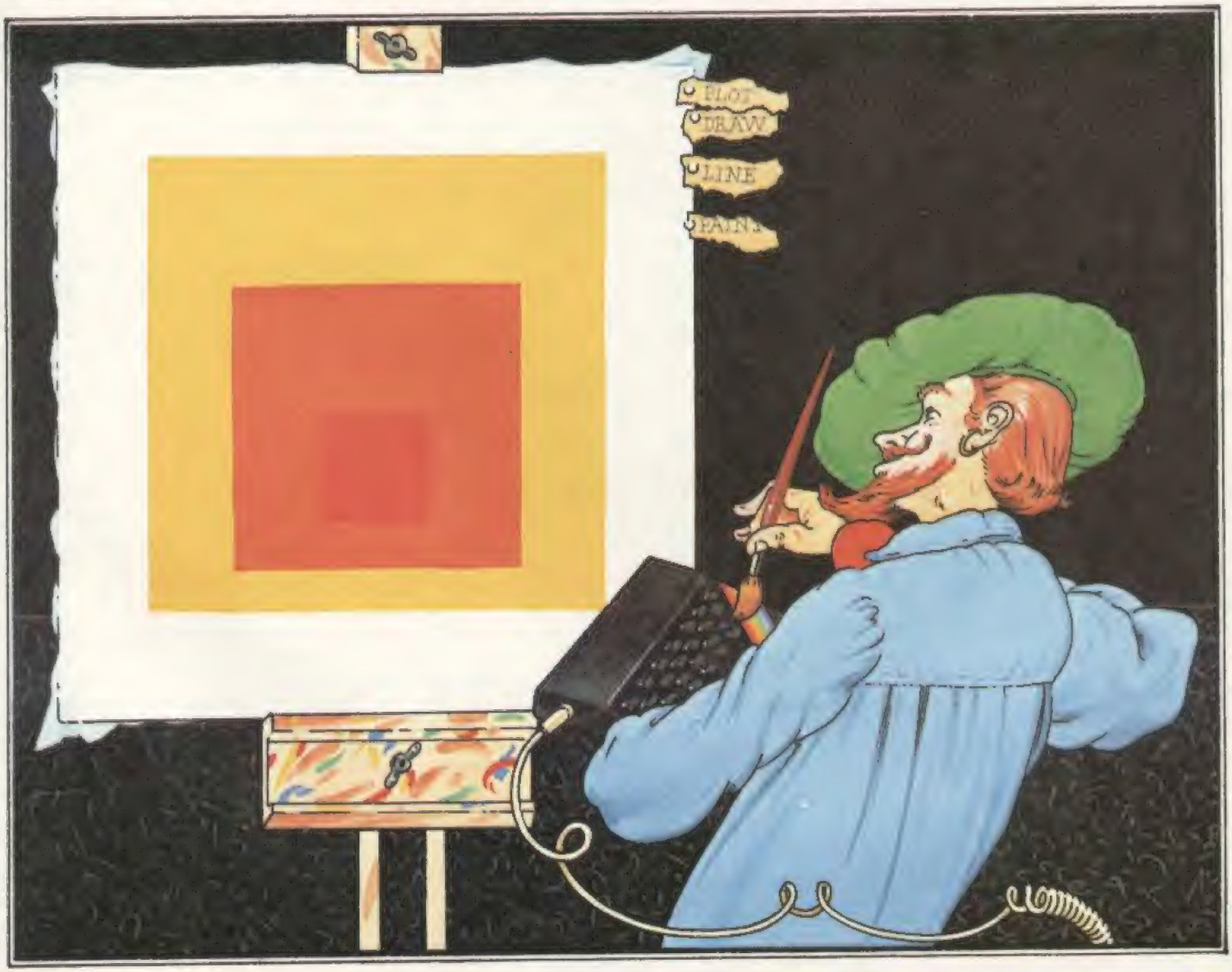
**Alguns comandos simples e muita imaginação: isso é tudo que você precisa para produzir imagens sensacionais na tela do computador. Veja aqui por onde começar.**

Ligue um aparelho de TV ou um monitor de vídeo ao seu computador e a telinha ficará à sua disposição. Praticamente todos os microcomputadores pessoais permitem que você desenhe nela, acrescente cor onde e como desejar, etc. Em pouco tempo você estará fazendo desenhos realmentte complicados.

Os gráficos para computador evoluíram muito nos últimos anos e atualmente são utilizados em muitos tipos de aplicações. Gráficos e tabelas constituem um exemplo óbvio, mas você também já deve ter notado o aumento no número de imagens geradas por computadores em propagandas de TV, revistas e até mesmo em efeitos especiais de filmes. Na verdade, muitas das ilustrações de INPUT foram produzidas ou modificadas por um computador.

É claro que os computadores utilizados para esses tipos de imagem sofisticada são muito maiores do que o modesto micro que você tem em casa. Mas existe uma quantidade impressionante de coisas que você pode fazer em sua maquininha doméstica.

A maioria dos micros entende comandos simples, tais como **PLOT**, **DRAW**, **PAINT** e **INK**. Para usá-los com eficiência e facilidade, contudo, é preciso compreender bem como a tela gráfica do computador é organizada.



Praticamente todos os computadores com capacidade gráfica de alta resolução utilizam a mesma forma de organização da tela. Esta é dividida em um conjunto de elementos individuais de imagem chamados *pixels* (termo que deriva da abreviatura da expressão inglesa *picture element*). Cada pixel é um pequeno retângulo na tela, que pode ser "ligado" ou "desligado" individualmente, por diferentes comandos. Os pixels são dispostos em uma espécie de matriz retangular, como um sistema de coordenadas, e cada um deles tem um "endereço", que geralmente é um par de números: sua posição na tela em relação ao eixo horizontal e ao vertical.

O número de pixels na tela define o seu *grau de resolução gráfica*. Em alguns computadores, você pode escolher entre desenhos de alta resolução com poucos "pixels" e linhas finas, ou desenhos de baixa resolução, com pixels maiores e linhas grossas. Muitos computadores permitem também que você selecione a cor que o pixel terá na tela ao ser iluminado.

Assim, desenhar na tela é apenas uma questão de dizer ao computador quais pixels devem ser acendidos. Em poucos minutos, experimentando os comandos gráficos mais simples, você estará fazendo belos desenhos sozinho.

Alguns micros, como os compatíveis com o ZX-81, têm uma capacidade gráfica relativamente limitada, pois seus programas exploram a utilização de dois comandos, apenas: o **PLOT**, que simplesmente acende um ponto na tela, e o **UNPLOT**, encarregado de apagá-lo. Além disso, a resolução gráfica da tela é bastante pequena. No entanto, esses micros podem ser equipados, opcionalmente, com uma placa ou um cartucho para gráficos de alta resolução.

Já os microcomputadores das linhas TRS-Color, Apple II, Sinclair Spectrum e MSX dispõem de recursos bastante sofisticados para desenhar em alta resolução.

Além disso, os micros mais modernos contam com comandos gráficos adicionais que permitem criar figuras geométricas, como círculos e retângulos, e pintá-las com cores diversas.

Para esses computadores, existe o conceito de *cursor gráfico*, ou seja, um ponto imaginário na tela que indica as coordenadas finais do último ponto ou reta traçados.

## S

A imagem de alta resolução que o Spectrum produz na tela é de 256 pixels no sentido horizontal e 176 no sentido vertical.

Os pixels são numerados de uma maneira aparentemente estranha à primeira vista. Se você já utilizou as declarações **PRINT AT** antes, sabe que:

```
PRINT AT 10,15 ...
```

significa que o que deve ser impresso será colocado na tela em uma posição definida por 10 linhas, contadas a partir do alto da tela, e 15 colunas, contadas a partir da borda esquerda.

Entretanto, quando você emprega a instrução **PLOT** para acender um pixel, o primeiro parâmetro do comando fornece a distância a partir da margem esquerda do vídeo, e o segundo define a distância para cima, a partir do lado inferior da tela. Esse método, aparentemente em conflito com o anterior, segue a convenção adotada para o sistema de coordenadas ortogonais (cartesianas). Assim, por exemplo, o comando

```
PLOT 10,15
```

diz ao computador para colocar um sinal na tela distante 10 pixels da margem esquerda e 15 linhas contadas de baixo para cima. Equivale, portanto, a X = 10 e Y = 15. Se você fizer a experiência verá o ponto aparecer na parte inferior esquerda da tela. As posições horizontais são numeradas de 0 a 255, e as verticais, de 0 a 175. Para sentir melhor como funciona a numeração das posições de tela, tente digitar essas linhas, uma de cada vez:

```
PLOT 0,0
PLOT 0,175
PLOT 255,0
PLOT 255,175
```

Os quatro pontos vistos agora na tela marcam os limites da área na qual se pode desenhar alguma coisa. Tudo o que estiver fora desses limites cairá numa parte da tela conhecida como moldura ou borda (**BORDER**) (veja mais adiante). Você pode usar, neste caso, o comando **PLOT** com todas as cores disponíveis no Spectrum.

## COMO DESENHAR RETAS

O comando **DRAW** do Spectrum faz exatamente o que você espera: ele desenha uma linha reta.

Antes, porém, o computador tem que saber onde começar a desenhá-la, qual será o seu comprimento, e em que direção você quer orientá-la.

Para iniciar a reta, você pode utilizar uma posição qualquer na tela, conforme foi definido para o comando **PLOT**. Tente essas retas em ordem, sem limpar a tela entre elas:

```
PLOT 100,75
DRAW 50,0
```

Se você não fornecer uma nova coordenada na tela, por intermédio do comando **PLOT** (que pula para ela, sem desenhar nada), o computador continuará a desenhar a partir da última posição, ou seja, do ponto final da última reta traçada (que é onde está o cursor gráfico). Para comprovar isso, adicione as seguintes retas, uma de cada vez, às duas que você acabou de desenhar:

```
DRAW 0,50
DRAW -50,0
DRAW 0,50
```

Como você pode observar, é necessário empregar números positivos para desenhar para cima ou para a direita e números negativos para desenhar para baixo ou para a esquerda.

Para traçar uma reta inclinada, você deve seguir a mesma regra: primeiro, estabeleça quantas posições para a direita ou para a esquerda o cursor gráfico deve se movimentar, depois quantas posições para cima ou para baixo. Por exemplo, acrescente essa reta àquelas já traçadas acima:

```
DRAW 50,50
```

## CONSTRUA UM PROGRAMA

Eis aqui um programa mais interessante, utilizando o comando **DRAW**. Para que você possa ver o que está acontecendo, entre e execute as linhas dos programas abaixo:

```
10 INK 2
20 PLOT 0,175
30 LET t=255: LET r=-175:
LET b=-255: LET l=169
40 DRAW t,0: DRAW 0,r: DRAW b
,0: DRAW 0,l
```

Se você digitou o programa corretamente (e não cometeu erros, como utilizar o número 1 no lugar do L minúsculo, ou vice-versa), obterá um retângulo sem um dos lados. Em seguida, adicione ao programa as linhas abaixo:

```
50 LET t=t-6
60 LET r=r+12: LET b=b+12:
LET l=l-12
70 DRAW t,0: DRAW 0,r: DRAW b
,0: DRAW 0,l
```

Ao executar o programa, você notará que agora apareceu um outro retângulo menor, dentro do primeiro. Acrescente então o resto do programa:

```
80 LET t=t-6
90 GOTO 50
```

Veja o que acontece: cada vez que o seu programa executa o laço entre as linhas 50 e 90, deduz 12 de cada um dos números positivos. A variável **t** (topo), por exemplo, diminui 255 para 243, depois para 231... para 219... e assim por diante. Eventualmente, ele vai aquém do zero e se torna um número negativo — assim, a reta que está sendo controlada pelo laço muda de direção. Do mesmo modo, as variáveis inicialmente negativas (tais como **b**, para baixo) podem tornar-se positivas e também mudar de direção.

Para observar mais de perto esse processo, rode novamente o programa com a seguinte adição:

```
85 PAUSE 100
```

## LUZ E SOMBRA

O Spectrum não tem a declaração **COLOR**, usada para colorir fácil e rapidamente uma área de fundo.

Todavia, você pode "sombrear" o desenho, utilizando uma série de retas paralelas de uma só cor. Mas, como cada nova reta precisa ter seu próprio comando **PLOT**, os programas tendem a se tornar demasiado lentos.

Um modo de se contornar essa dificuldade é utilizar um laço **FOR...NEXT:** o programa seguinte desenha a parede direita de uma casa (figura 2):

```
20 FOR h=56 TO 100 STEP 3
30 PLOT 160,h
40 DRAW 23,12
50 NEXT h
60 PLOT 183,110 : DRAW 0,-9:
 : DRAW   -23,-13: DRAW 0,100
```

Como isso funciona? Cada vez que o laço é executado, o programa desenha uma linha de 23 pixels, inclinando-a de 12 pixels, da esquerda para a direita. A cada nova linha, ele inicia 160 pixels à esquerda.

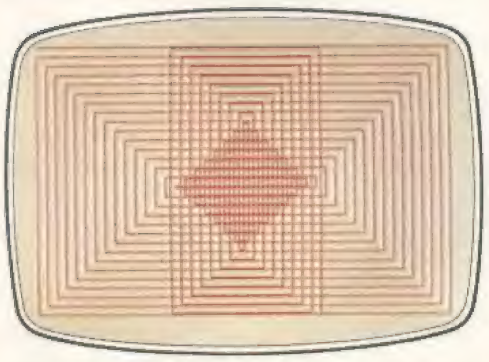


**Fig. 1: Um padrão de gráfico para o Spectrum.**

**Fig. 2: A casa em perspectiva.**

Mas a cada "viagem" do laço a linha 20 acrescenta 3 à variável **h**, que representa a altura da posição **PLOT**. Assim, o início de cada linha será três pixels mais alto que o da anterior.

Acrescente as próximas linhas e sua casa passará a ter a maior parte do telhado:

```
 70 LET r=100
 80 FOR p=140 TO 150
 90 PLOT r,p
100 DRAW 69,-44
110 LET r=r+2
120 NEXT p
```

Existem aqui duas variáveis, **p** e **r**. Isso se deve a que as posições **PLOT** movimentam-se não apenas para cima mas também para a direita.

Observe a utilização de números negativos na linha 100 para fazer o declive do telhado para baixo. As próximas linhas completam o contorno da casa:

```
130 PLOT 100,140
140 DRAW -60,-40
150 PLOT 46,103
160 DRAW 0,-96
170 DRAW 113,-7
```

Agora você pode terminar a casa, uma porta, as janelas e uma chaminé com o auxílio dos comandos **PLOT** e **DRAW**. Se você decidir copiar a casa que aparece na figura 1, eis aqui algumas sugestões:

O triângulo do revestimento abaixo do telhado requer três variáveis. A primeira define a altura onde cada linha começa. A segunda controla a distância, à direita, de onde ela parte. A terceira, finalmente, determina o comprimento da linha.

As janelas do andar superior são traçadas sobre o revestimento, depois que este é desenhado. Para fazer isso, você precisará jogar espaços em branco, usando o **PRINT AT**, nos pontos adequados. Em seguida, trace o contorno das janelas, utilizando **PLOT** e **DRAW** da maneira usual.

## COMO DESENHAR CÍRCULOS

A declaração **CIRCLE** do Spectrum também se comporta de modo previsível: ela desenha um círculo.

É necessário não esquecer que os primeiros dois números após uma declaração **CIRCLE** definem as coordenadas do centro do círculo — o qual não é traçado —, enquanto o terceiro número fornece o seu raio em pixels. Assim a linha:

```
CIRCLE 127,87,50
```

... desenha um círculo de 100 pixels de diâmetro (50 de raio) no meio da tela (posição x = 127 e y = 87).

No programa da casa, por exemplo, você pode "instalar" uma janela redonda na porta da frente, utilizando a declaração **CIRCLE**.

Se você seguir um **CIRCLE** com uma declaração **DRAW**, a linha que você desenhar começará a partir do ponto onde o círculo termina.

## CORES NO SPECTRUM

O Spectrum tem uma gama de oito cores, identificadas por rótulos acima das teclas numéricas de 0 a 7, que são utilizadas para chamá-las quando isso se faz necessário. O comando **BORDER** (bordas) controla a área externa da tela, sobre a qual você não pode desenhar nem imprimir nada.

Os outros comandos de cores podem ser utilizados de duas maneiras bem diversas. Se você rodar um programa com, por exemplo,

```
10 BORDER 2:PAPER 2:INK 7
```

tudo o que se seguir no programa, depois que essa linha for executada, será impresso em branco **(INK 7)**, sobre uma tela de fundo vermelho **(PAPER 2)**. Tal situação só será alterada quando você substituir essa linha por outra que modifique os comandos de cor. Note que a moldura é da mesma cor que o fundo da tela **(BORDER 2)**, mas você poderia ter especificado outra cor para ela. Até mesmo as listagens dos programas aparecerão nas cores indicadas, o que pode dificultar a sua leitura. Se, por outro lado, você iniciar com:

```
10 PRINT PAPER 2:INK 7:AT
   15,10; "*"
```

então, apenas a pequena área da tela onde o asterisco entre aspas aparece ficará vermelha.

Observe que no primeiro exemplo as declarações de cor são seguidas por dois pontos, ao passo que no segundo exem-

plo cada uma delas é separada por um ponto e vírgula.

Você pode incorporar essas declarações de cor a qualquer linha do programa, utilizando **PLOT**, **DRAW** ou **CIRCLE**. Tente essas linhas limpando a tela (comando **CLS**) entre as linhas:

```
PLOT PAPER 1:INK 7:125,87
```

(Dá para ver o ponto?)

```
PLOT 125,87:DRAW INK 2;50,50
PLOT 125,87:DRAW PAPER 1; INK
     7;50,50
CIRCLE INK 4;127,87,50
```

e até mesmo esta:

```
CIRCLE PAPER 2;INK 6;125,87,50
:DRAW
PAPER 3;INK 7;75,0
```

Esta última combinação pode ser utilizada para produzir alguns efeitos espetaculares.

O último dos comandos gráficos do Spectrum tratados neste artigo — o **FLASH** — é igualmente espetacular, e muito utilizado em programação de jogos. O que ele faz é reverter rapidamente as cores da tinta e do papel.

O **FLASH** deve ser seguido por um número — tanto o 1 para ligar, como o 0 para desligar; eis aqui um exemplo de sua utilização:

```
10 PAUSE 0
20 IF INKEY$="" THEN  GOTO 30
30 FOR n=10 TO 23: SOUND .015
,n: NEXT n
40 BORDER 2: PAPER 2: INK 6:
FLASH 1
50 PRINT "NAO TOQUE !"
60 PRINT "ESTE COMPUTADOR E P
ERIGOSO!"
70 PRINT
80 GOTO 30
```

Quando você tiver entrado esse programa, rode-o. Então pressione qualquer tecla... e saia de perto!

## S

O ZX-81 utiliza o comando **PLOT** de um modo parecido ao do Spectrum: **PLOT**, seguido por dois números que controlam a posição do pixel. Mas existe uma diferença: os pixels do ZX-81 são dezesseis vezes maiores do que os pixels do Spectrum, pois neste existem apenas quatro pixels em cada espaço de caractere, enquanto que o Spectrum tem 64 pixels em um espaço de caractere.

Como resultado, você não conseguirá fazer desenhos ou padrões tão detalhados; mas tornará cada pixel mais visível.

O programa a seguir traça um padrão gráfico na tela. Como ele recomeça cada vez que vai para a linha 70, você deverá pressionar a tecla <**BREAK**> quando quiser parar:

```
10 LET X=INT (RND*32)
20 LET Y=INT (RND*24)
30 PLOT X,Y
40 PLOT 63-X,Y
50 PLOT X,43-Y
60 PLOT 63-X,43-Y
70 GOTO 10
```

O ZX-81 não tem o comando **DRAW**, mas você pode utilizar a instrução **PLOT** para compor uma reta com pontos. Você poderá ver como isso funciona com o programa seguinte, o qual desenha um certo número de quadrados utilizando dois laços **FOR...NEXT**.

```
10 FOR N=0 TO 20 STEP 2
20 FOR M=N TO 43-N
30 PLOT M,N
40 PLOT M,43-N
50 PLOT N,M
60 PLOT 43-N,M
70 NEXT M
80 NEXT N
```

E de um modo parecido, você pode imitar o comando **CIRCLE**, empregando a seguinte rotina:

```
10 FOR N=0 TO 2*PI STEP .1
20 PLOT 32+20*SIN N,22+20*COS N

30 NEXT N
```

Ela utiliza as duas funções matemáticas — o seno (**SIN**) e o cosseno (**COS**) — para calcular as coordenadas dos pixels que compõem o círculo. Elas serão explicadas em detalhes em um artigo posterior, mas, no momento, você pode experimentar com o tamanho do círculo, modificando os dois números 20. O número máximo que você pode ter é 22 (maior que isso não caberá na tela). Outra maneira é modificar a posição do círculo. No momento, seu centro está em 32,22 (você pode ver essas coordenadas na linha 20). Não se esqueça de fazer o círculo menor se você movê-lo para perto da borda da tela.

Os microcomputadores compatíveis com Apple II também têm recursos razoáveis para a elaboração de desenhos em alta resolução, em cores.

O Apple tem três modalidades de tela: a tela de texto (invocada pelo comando **TEXT** e limpada pelo comando **HOME**); a tela gráfica de baixa resolução (que tem 40 pixels na direção horizontal e 40 pixels na direção vertical, usando dezesseis cores); e a tela gráfica de alta resolução (que tem 280 pixels na direção horizontal e 192 na direção vertical, com seis cores). O comando para ligar a tela gráfica de baixa resolução é o **GR**; e, para a tela de alta resolução, o **HGR**.

Existe um conjunto diferente, embora parecido, de comandos gráficos para os modos de alta e de baixa resolução. Para o modo de baixa resolução temos os seguintes: **COLOR**, **PLOT**, **HLIN** e **VLIN**. Para a alta resolução, os comandos que mostraremos nesta lição são o **HCOLOR** e o **HPLOT**.

Além disso, existem outros comandos mais sofisticados para o modo de alta resolução, tais como **DRAW**, **ROTATE**, **SCALE**, etc., que serão tratados em uma lição futura.

Os micros da linha Apple II têm a limitação de não permitir a exibição simultânea de letras e de gráficos de baixa e de alta resolução na mesma tela. Só o modelo nacional TK-2000, da Microdigital, tem essa capacidade.

### LINHAS RETAS E COR

Para demonstrar a utilização dos comandos gráficos de alta resolução no Apple II e no TK-2000, vamos fazer um desenho bem simples, passo a passo: um aparelho de TV em que aparecem uma série de pontos coloridos, distribuídos aleatoriamente na tela. O programa começa com as linhas a seguir, que desenham a caixa do televisor:

```
10   HGR2
20   HCOLOR= 3
30   HPLOT 10,75 TO 200,75
40   HPLOT 200,75 TO 200,190
50   HPLOT  TO 10,190
60   HPLOT  TO 10,75
```

A linha 10 define o tipo de tela gráfica a ser utilizada, com o comando **HGR2**. O **HGR** significa que a tela será de alta resolução (*High-resolution GRaphics*), e o número 2 diz ao computador que a tela será inteiramente tomada pelos gráficos (se não houvesse esse 2, o computador reservaria automaticamente cinco linhas de texto na parte de baixo da tela). Sem esse comando, o computador não aceitaria os comandos gráficos subseqüentes.

A linha 20 define a cor a ser usada para o traço em alta resolução. É usado o comando **HCOLOR**, seguido do número da cor desejada. O Apple não tem um comando para definir a cor de fundo, como é o caso do MSX e do TRS-Color. Se for necessário ter uma cor de fundo diferente do preto (a normal da tela), usa-se uma sucessão de li-

nhas paralelas da mesma cor (essa manobra, porém, leva muito tempo).

As linhas 30 a 60 traçam um retângulo na tela, com o canto superior esquerdo em (10,75) e o canto inferior direito em (200,190). O comando utilizado é o **HPLOT**. Esse comando pode ser usado de várias maneiras.

A variante do **HPLOT** mostrada nesse segmento de programa serve para traçar retas. Os parâmetros básicos para o **HPLOT** são conjuntos de dois números ou expressões numéricas, separados por uma vírgula. Tais parâmetros indicam a posição em que o cursor gráfico deverá ser colocado na tela de alta resolução. Por exemplo: **HPLOT 10,75...** diz ao computador para começar a traçar uma reta em alta resolução, nas coordenadas X = 10 e Y = 75 (ou seja, 10 pixels distantes da margem esquerda da tela, e 75 pixels abaixo do topo).

Para traçar uma reta, usa-se o comando do **HPLOT** seguido da palavra **TO**. Por exemplo, na linha 30 do programa, traçamos uma reta começando no ponto (10,75), e indo até o ponto (200,75) — portanto, paralela ao eixo horizontal — usando o comando:

```
HPLOT 10,75 TO 200,75
```

O cursor gráfico agora fica no último ponto da tela depois do **TO:** no exemplo acima, em (200,75). Se quisermos traçar uma reta a partir daí, não é necessário especificar um novo começo para ela. Basta usar o comando assim:

```
HPLOT TO 200,190
```

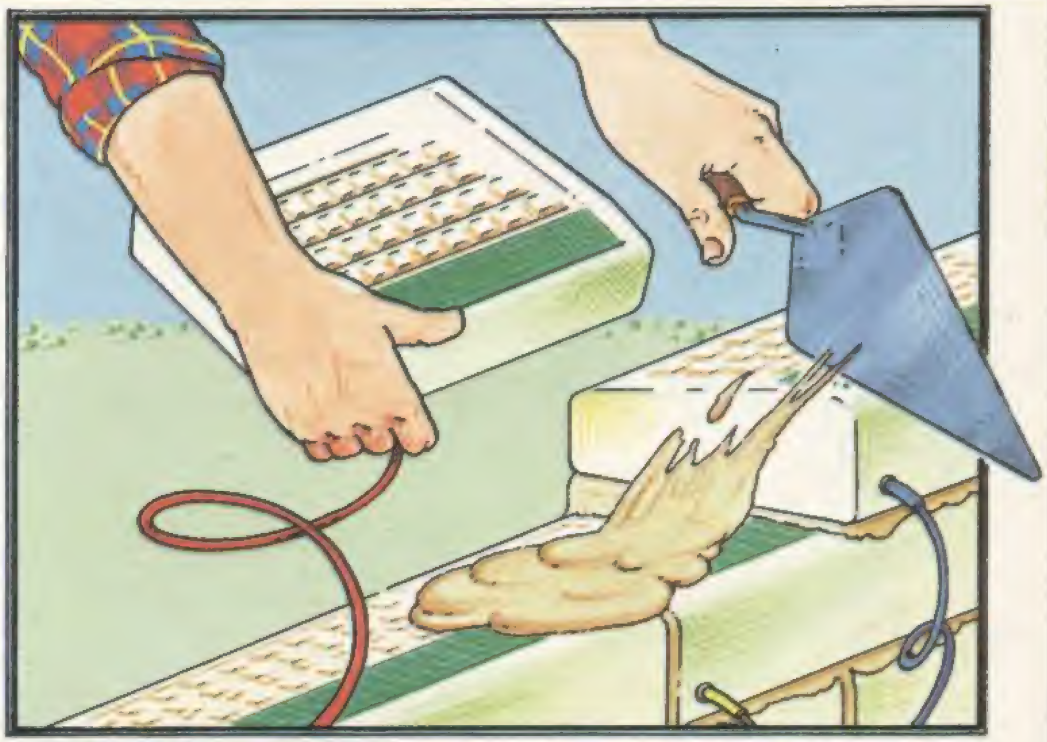
Essa nova forma vai traçar uma linha de (200,75) a (200,190) — uma reta vertical, portanto.

A linha 300 simplesmente "congela" a tela gráfica, impedindo que o fim do programa a cancele.

Vamos traçar agora o restante do televisor, adicionando as linhas abaixo:

```
70  HPLOT 25,85 TO 155,85 TO 15
5,180 TO 25,180 TO 25,85
80  HPLOT 10,75 TO 45,50
90  HPLOT 200,75 TO 235,50
100  HPLOT 200,190 TO 235,165
110  HPLOT 45,50 TO 235,50 TO 2
35,165
120  HPLOT 190,60 TO 230,0
130  HPLOT 190,60 TO 150,0
140  HPLOT 175,85 TO 179,85 TO
179,89 TO 175,89
150  HPLOT 175,105 TO 179,105 T
O 179,109 TO 175,109
160  HPLOT 175,135 TO 179,135 T
O 179,139 TO 175,139
170  HPLOT 175,155 TO 179,155 T
O 179,159 TO 175,159
```

A linha 70 traça um outro retângulo dentro do primeiro. Note que podemos utilizar uma série de cláusulas **TO**, dentro da mesma linha com o **HPLOT**. A linha 70 funciona de modo idêntico às quatro linhas dos números 30 a 60.



As linhas 80 e 90 servem para desenhar a antena. Note que linhas inclinadas são traçadas exatamente com a mesma técnica do **HPLOT**. As linhas restantes, de 100 a 170, servem para traçar o restante do desenho do televisor, e dar perspectiva a ele.

Finalmente, para fazer aparecer um "chuvisco" colorido na tela do televisor que está sendo esboçado no vídeo do seu computador, adicione as linhas:

```
180 X =  INT ( RND (1) * 130) +
 26
190 Y =  INT ( RND (1) * 95) +
86
200  COLOR=  RND (1) * 10
210  HPLOT X,Y
220  GOTO 180
```

As coordenadas dos pontos aleatórios são definidas nas linhas 180 e 190 e são armazenadas nas variáveis X e Y. Note que a expressão aritmética sorteia, usando a função **RND**, um número inteiro dentro dos limites da tela.

O **HPLOT** mostrado aqui é mais simples, pois é seguido apenas por dois números ou duas expressões numéricas, separadas entre si por uma vírgula. Esses dois parâmetros indicam a posição em que o cursor gráfico deverá ser colocado na tela de alta resolução.

```
HPLOT X,Y
```

diz ao computador para traçar um ponto em alta resolução, nas coordenadas X e Y.

Finalmente, a linha 220 gera novos pontos aleatórios em um laço infinito. Para interrompê-lo, use as teclas **<CTRL><C>**.

## CÍRCULOS NA TELA

Os micros da linha Apple não têm um comando específico para traçar círculos, mas você pode imitar o comando **CIRCLE** utilizando a seguinte rotina:

```
10  HOME : INPUT "COR:";C
12  INPUT "CENTRO (X,Y):";XC,YC
14  INPUT "RAIO:";R
15  HCOLOR= C
16  HGR
17  HPLOT XC,YC + R
20  FOR N = 0 TO 6.2856 STEP .1
```

**Formas simples como esta televisão podem ser desenhadas no Apple II e no TK-2000.**

```
30  HPLOT  TO XC + R *  SIN (N)
,YC + R *  COS (N)
40  NEXT N
```

Essa rotina utiliza as duas funções matemáticas — o seno (**SIN**) e o cosseno (**COS**) — para calcular as coordenadas de cada pixel que compõe o círculo. Tais funções serão melhor explicadas em uma futura lição; no momento, porém, você pode experimentar com o tamanho do círculo, modificando os dois números 20 na linha 20. O número máximo que se pode ter é 22 (a tela não comporta um valor mais alto). Você pode também modificar a posição do círculo. Seu centro está em 32,22; observe essas coordenadas na linha 20. Não se esqueça de fazer o círculo menor se você movê-lo para perto da borda da tela.

## T

Assim que você liga o computador, ele exibe a tela de texto. Nela você pode mostrar todos os caracteres do teclado, mais os caracteres gráficos da ROM, já utilizados antes.

Se você quiser desenhar qualquer coisa mais sofisticada do que aquilo que esses simples gráficos permitem, deve empregar os gráficos de alta resolução da máquina. Estes são obtidos em uma tela completamente diferente da tela de texto — pois, nos computadores compatíveis com o TRS-Color, não é possível colocar-se, ao mesmo tempo, texto e gráficos de alta resolução na mesma tela.

## RETAS CRUZADAS

Eis aqui um programa que desenha duas retas cruzadas na tela, utilizando a tela gráfica em modo 3:

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 LINE (0,191)-(255,0),PSET
60 LINE (0,0)-(255,191),PSET
70 GOTO 70
```

O programa começa especificando a modalidade gráfica e a página (parte da memória do computador) em que você deseja colocar a informação gráfica. A linha 20 declara isso com **PMODE 3,1**. Essa informação diz à máquina para desenhar em modalidade de gráficos 3 e para armazenar qualquer coisa que você desenhar na tela na página de memória gráfica número 1. Se você quiser desenhar apenas uma tela cheia de gráficos será mais fácil especificar a página 1.

A linha 30 limpa a tela de alta resolução. **PCLS** é o comando para alta resolução, equivalente ao **CLS**.

Para "ligar" a tela de alta resolução o programa utiliza o comando **SCREEN 1,0**. O conjunto de cores é escolhido colocando-se 0 como o segundo no comando **SCREEN**: esse conjunto consiste de verde, amarelo, azul e vermelho. Se, ao invés disso, você tivesse colocado 1, o conjunto de cores seria branco, azul ciano, magenta e laranja.

Esse programa desenha linhas em verde e vermelho, que são as cores predefinidas (são as normalmente utilizadas pelo computador, se você não especificar nenhuma outra). Para utilizar qualquer outra combinação do conjunto de cores você deverá utilizar o comando **COLOR**. Isso será explicado em detalhes, futuramente.

A primeira reta é desenhada pelo comando **LINE** na linha 50. Os números entre parênteses dizem ao computador em que pontos você quer que a linha comece e termine. A tela de alta resolução é dividida em 256 pixels por 192 — no sentido vertical, eles são numerados de 0 a 191, a partir do alto; no sentido horizontal, de 0 a 255, a partir da esquerda. Note que a convenção adotada no TRS-Color para as coordenadas cartesianas é o oposto ao normal.

Portanto, **LINE (0,191)-(255,0)** traça uma reta com a origem em X = 0 e Y = 191 e fim em X = 255 e Y = 0.

Finalmente, **PSET** sozinho, no comando **LINE**, diz à máquina para desenhar a cor de maior número do conjunto de cores — neste caso, o vermelho.

A linha 60 desenha a próxima reta. Isto funciona exatamente do mesmo modo como na linha 50, mas os pontos iniciais e finais são definidos novamente.

Para evitar que o computador retorne automaticamente para a tela de texto, uma vez que a tela de alta resolução tenha sido desenhada, estabelecemos um laço sem fim: é o que a linha 70 faz, com o comando **GOTO 70**. Se o computador ligasse novamente a tela de texto, não seria possível ver o que foi desenhado na tela de alta resolução.

## CÍRCULOS NO TRS-COLOR

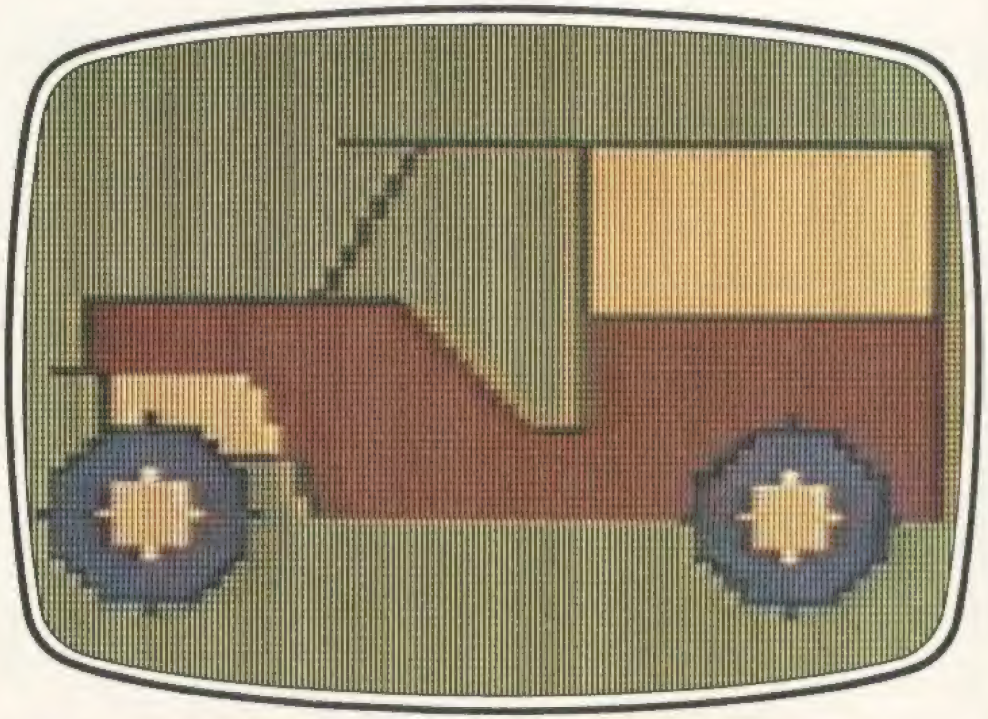
É muito fácil desenhar círculos no TRS-Color, pois o seu BASIC possui um comando **CIRCLE**.

Este programa desenhará três círculos com tamanhos diferentes, cada um com uma cor diferente.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 CIRCLE (70,95),10,2
60 CIRCLE (118,95),20,3
70 CIRCLE (184,95),30,4
80 GOTO 80
```

A modalidade gráfica é escolhida como no programa anterior, assim como o conjunto de cores para os círculos.

O comando **CIRCLE** nas linhas 50 a 70 tem quatro elementos. O primeiro e



**O desenho do jipe torna-se fácil com o comando *PAINT* do TRS-Color.**

o segundo números definem as coordenadas do centro do círculo, o terceiro é o seu raio, em pixels, e o quarto é a cor. Assim, o círculo desenhado pela linha 50 tem o seu centro em (70,95), um raio de 10 pixels e está colorido de amarelo (Cor 2).

O círculo desenhado pela linha 60 é azul, com raio de 20 pixels, e com centro em (118,95). O último círculo, desenhado pela linha 70, é vermelho, com raio de 30 pixels e com centro em (184,95).

Finalmente, a linha 80 é um laço que impede que o programa pare e desligue a tela de alta resolução.

## DESENHE UM JIPE

Agora você possui os ingredientes para desenhar muitas coisas diferentes. Digite e rode o programa abaixo e você verá o esboço da carroçaria de um jipe, ainda sem os detalhes internos mostrados na figura 3.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 LINE (108,64)-(188,64),PSET
60 LINE-(188,116),PSET
70 LINE-(104,116),PSET
80 LINE-(96,96),PSET
90 LINE-(70,96),PSET
110 LINE-(74,96),PSET
120 LINE-(74,86),PSET
130 LINE-(114,86),PSET
140 LINE-(132,104),PSET
150 LINE-(140,104),PSET
160 LINE-(140,64),PSET
170 LINE(76,96)-(76,108),PSET
180 LINE-(100,108),PSET
320 GOTO 320
```

Assim como antes, a modalidade escolhida é a de número 3. A linha 50 começa traçando uma linha vermelha de (108,64) a(188,64). Para continuar o desenho a partir do mesmo ponto, não se deve dizer ao computador por onde começar — ele apenas prossegue de onde estiver no momento. As linhas 60 a 160, portanto, apenas definem os finais de linhas sucessivas.

A linha 170, no entanto, desenha uma linha a partir de um novo ponto inicial; assim tanto o início como o fim devem ser definidos.

Acrescente agora algumas linhas para definir o pára-brisa, as rodas e as calotas:

```
200 LINE(140,88)-(188,88),PSET
220 LINE(118,64)-(104,86),PSET
230 CIRCLE(82,116),14,3
250 CIRCLE(82,116),6,2
270 CIRCLE(168,116),14,3
290 CIRCLE(168,116),6,2
```

A linha 200 desenha a janela, enquanto o pára-brisa é determinado pela linha 220. O comando **CIRCLE** é utilizado pelas linhas 230 e 270 para definir as rodas e pelas linhas 250 e 290 para desenhar as calotas.

## PINTE UMA ÁREA

Agora que o desenho está feito, é fácil colorir o seu jipe: você nem precisa de uma pistola de pintura, porque a máquina faz isso através do comando **PAINT**.

O comando **PAINT** preenche com uma cor específica qualquer forma que você tenha desenhado na tela. Ele também possui quatro elementos. Os dois primeiros, como você já deve ter adivinhado, marcam o local onde a pintura deve começar. O terceiro é o código da cor a ser utilizada. E o quarto é o código de cor da borda, ou da linha onde a pintura deve acabar.

Acrescente essas linhas e você verá, na prática, como fica o programa:

```
190 PAINT(90,100),2,4
210 PAINT(120,110),4,4
240 PAINT(82,116),3,3
260 PAINT(82,116),2,2
280 PAINT(168,116),3,3
300 PAINT(168,116),2,2
310 PAINT(180,80),2,4
```

Rode agora o programa: como num passe de mágica, o jipe se cobrirá de cores.

Não é possível colocar ao final do programa todas as linhas com **PAINT**, por uma razão muito importante:

A área que você quer colorir deve ser fechada completamente por uma série de retas da mesma cor (a cor da borda), ou pela borda da tela. Qualquer linha de cor diferente que estiver localizada no meio da figura geométrica a ser preenchida será pintada por cima.

As cores disponíveis no **PMODE 3,1** são: verde (1), amarelo (2), azul (3) e vermelho (4). Se você quiser pintar o jipe de modo diferente, tudo o que tem a fazer é modificar o número de cor da pintura nas linhas **PAINT** correspondentes.

Você pode, se quiser, acrescentar outras partes ao jipe, equipando-o com novos elementos, como uma roda de direção, por exemplo. Para isso, empregue os comandos **LINE** e **CIRCLE**.

Assim que você ligar o computador, ele exibirá a tela de texto, na qual podem ser mostrados todos os caracteres do teclado, mais os caracteres do gráfico da ROM. Estes últimos já foram utilizados em programas anteriores.

Para desenhar figuras mais sofisticadas do que aquelas que vimos até agora, é necessário utilizar os gráficos de alta resolução da máquina. Estes são obtidos em uma tela completamente diferente da tela de texto, já que os computadores da linha MSX não permitem que sejam colocados, ao mesmo tempo, texto e gráficos de alta resolução numa única tela.

## VOLTANDO À LINHA RETA

Eis aqui um programa que desenha duas retas cruzadas no vídeo, utilizando a tela gráfica em modo 2:

```
10 SCREEN 2
20 LINE(0,191)-(255,0),15
30 LINE(0,0)-(255,191),1
40 GOTO40
```

O programa começa especificando a modalidade gráfica e a página (parte da memória do computador) na qual você deseja colocar a informação gráfica. A linha 20 declara isso com **SCREEN 2**, dizendo à máquina para desenhar em modalidade de gráficos 2.

A primeira reta é desenhada pelo comando **LINE**, na linha 20. Os números entre os parênteses dizem ao computador onde a linha deve começar e terminar. A tela de alta resolução é dividida em 256 pixels por 192: no sentido vertical eles são numerados de 0 a 191, a partir do alto; no sentido horizontal, de 0 a 255, a partir da esquerda.

Portanto, **LINE (0,191)-(255,0)** traça uma reta com origem em X = 0 e Y = 191 e fim em X = 255 e Y = 0.

O número no final do comando **LINE** diz à máquina que cor ela deve empregar (neste caso, existem dezesseis cores, numeradas de 0 a 15).

A linha 30 desenha a próxima reta. Isso funciona do mesmo modo que na linha 20, mas os pontos iniciais e finais são definidos novamente.

Para evitar que o micro retorne à tela de texto uma vez que a tela de alta resolução tenha sido desenhada, estabelecemos um laço sem fim: é o que a linha 40 faz, com **GOTO 40**. Se o computador religasse a tela de texto, você não seria capaz de ver o que foi desenhado na tela de alta resolução.

## CÍRCULOS NO MSX

Assim como no TRS-Color, é fácil desenhar círculos no MSX, pois o seu

BASIC conta com um comando chamado **CIRCLE**. Este programa desenha três círculos com tamanhos diferentes, cada um de uma cor:

```
10 COLOR 4,15
20 SCREEN2
30 CIRCLE(70,95),10,2
40 CIRCLE(118,95),20,4
50 CIRCLE(184,95),30,8
60 GOTO 60
```

**Qual a diferença entre gráficos de baixa, média e alta resolução?**

O grau de resolução gráfica a ser obtido em um computador depende do número de pixels que se pode endereçar individualmente na tela. O pixel (que vem da abreviação, em inglês, de *picture element*) é o pontinho gráfico que um comando BASIC acende ou apaga. Naturalmente, como a tela tem um tamanho limitado, a resolução é proporcional às dimensões físicas de cada pixel.

Do ponto de vista técnico, a alta resolução gráfica é obtida quando há mais de 100 000 pixels por tela (por exemplo, o micro de tipo IBM-PC tem 600 pixels na horizontal por 200 na vertical). A resolução média ficaria com algo entre 30 000 e 100 000 pixels; a essa faixa pertencem o Sinclair Spectrum, o TRS-Color, o Apple II e o TK-2000. Já a resolução baixa tem de 2 500 a 10 000 pixels por tela (por exemplo, o TRS-80 e o ZX-81). Entretanto, muitos fabricantes (como os da linha Apple II) chamam de alta resolução os gráficos tecnicamente definidos como de média resolução.

**O TRS-80 pode ser usado para desenhar em BASIC?**

Sim, embora os seus gráficos apareçam apenas em baixa resolução (128 por 48 pixels, em preto e branco). O comando a ser utilizado para "acender" um pixel na tela é o **SET**. Esse comando deve ser seguido de dois parâmetros ou expressões numéricas: o primeiro **(X)** indica a posição horizontal do pixel e o segundo **(Y)**, a sua posição vertical. Ele equivale, portanto, ao comando **PLOT** dos micros da linha Sinclair. Existem ainda os comandos gráficos **RESET** (para "apagar" o pixel aceso) e **POINT** (uma função que informa se um determinado pixel está aceso ou apagado).

A modalidade gráfica é escolhida como no programa anterior. Só que a cor de fundo da tela é definida antes, com o comando **COLOR**. O primeiro número é a cor do caractere (cor de frente), e o segundo, a cor de fundo.

O comando **CIRCLE** nas linhas 30 a 50 tem quatro elementos. O primeiro e o segundo números definem as coordenadas do centro do círculo, o terceiro é o seu raio, em pixels, e o quarto é a cor. Assim, o círculo desenhado pela linha 30 tem o seu centro em (70,95), um raio de 10 pixels e é colorido de amarelo (cor 2).

O círculo desenhado pela linha 40 é azul, com um raio de 20 pixels, e centro em (118,95). O último círculo, desenhado pela linha 50, é vermelho, com raio de 30 pixels e centro em (184,95). A linha 60 é um laço que impede que o programa pare e desligue a tela de alta resolução.

## DESENHE O JIPE NO MSX

Como você já deve ter percebido, algumas seções de nosso texto repetem seções anteriores. Isso se deve à similaridade dos procedimentos nos diversos micros. Agora, digite e rode o programa abaixo.

```
10 COLOR 4,15
20 SCREEN2
30 LINE(108,64)-(188,64),10
40 LINE -(188,88),10:LINE-(188,
116)
50 LINE -(104,116)
60 LINE-(96,96)
70 LINE-(70,96)
80 LINE-(74,96)
90 LINE-(74,86)
100 LINE-(114,86)
110 LINE-(132,104)
120 LINE-(140,104)
130 LINE-(140,64),10
140 LINE (76,96)-(76,108)
150 LINE-(100,108)
160 LINE(140,88)-(188,88),10
290 GOTO 290
```

Isso mesmo, ele recria o jipe, agora em outra máquina. Assim como antes, a modalidade escolhida é a 2. A linha 30 desenha uma linha vermelha de (108,64) a (188,64). Como da vez anterior, você não precisa dizer ao computador onde começar a desenhar: ele apenas prossegue de onde estiver no momento. As linhas 40 a 160, portanto, apenas definem os finais de linhas sucessivas.

A linha 150, no entanto, desenha uma linha a partir de um novo ponto inicial; assim, tanto o início como o fim devem ser definidos.

Acrescente agora algumas linhas para o pára-brisa, as rodas e as calotas:

```
180 LINE (140,89)-(188,89)
200 LINE(118,64)-(104,86)
210 CIRCLE(82,116),14,3
230 CIRCLE(82,116),6,2
250 CIRCLE(168,116),14,3
270 CIRCLE(168,116),6,2
```

As linhas 170 e 180 desenham a janela, e a linha 200, o pára-brisa. O comando **CIRCLE** é utilizado pelas linhas 210 e 250 para definir as rodas, e pelas linhas 230 e 270 para desenhar as calotas.

## PINTE O SEU JIPE

Para pintar o seu jipe, você não precisa de uma pistola de pintura: a máquina faz isso para você por meio do comando **PAINT**.

O comando **PAINT** preenche com uma cor específica qualquer forma que você tenha desenhado na tela. Ele também possui quatro elementos. Os dois primeiros, como você já deve ter adivinhado, marcam o local onde a pintura deve começar. O terceiro é o código da cor a ser empregada. O quarto é o código de cor da borda, ou da linha, onde a pintura deve acabar.

Acrescente essas linhas ao programa e você verá, na prática, como fica o programa:

```
170 PAINT (142,66),10,10
190 PAINT(77,99),4,4
220 PAINT(82,116),3,3
240 PAINT(82,116),2,2
260 PAINT(168,116),3,3
280 PAINT(168,116),2,2
```

Rode agora o programa e você verá cada seção do jipe sendo preenchida com belas cores.

Não é possível colocar todas as linhas com **PAINT** no final do programa, por uma razão muito importante:

A área que você pretende colorir deve ser fechada completamente por uma série de retas da mesma cor (a cor da borda), ou pela borda da tela.

Qualquer linha de cor diferente que estiver localizada no meio da figura geométrica a ser preenchida deve ser pintada por cima.

As cores disponíveis no **PMODE 3,1** são: verde (1), amarelo (2), azul (3) e vermelho (4). Mas você pode querer pintar o jipe de modo diferente; nesse caso, tudo o que tem a fazer é modificar o número de cor da pintura nas linhas **PAINT** correspondentes.

Outras partes, contudo, podem ser incorporadas ao nosso jipe (uma roda de direção, por exemplo, ou outro equipamento qualquer). Para isso, utilize os comandos **LINE** e **CIRCLE**.

## CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

| LINHA | FABRICANTE | MODELO |
|---|---|---|
| Apple II + | Appletronica | Thor 2010 |
| Apple II + | CCE | MC-4000 Exato |
| Apple II + | CPA | Absolutus |
| Apple II + | CPA | Polaris |
| Apple II + | Digitus | DGT-AP |
| Apple II + | Dismac | D-8100 |
| Apple II + | ENIAC | ENIAC II |
| Apple II + | Franklin | Franklin |
| Apple II + | Houston | Houston AP |
| Apple II + | Magnex | DM II |
| Apple II + | Maxitronica | MX-2001 |
| Apple II + | Maxitronica | MX-48 |
| Apple II + | Maxitronica | MX-64 |
| Apple II + | Maxitronica | Maxitronic I |
| Apple II + | Microcraft | Craf II Plus |
| Apple II + | Milmar | Apple II Plus |
| Apple II + | Milmar | Apple Master |
| Apple II + | Milmar | Apple Senior |
| Apple II + | Omega | MC-400 |
| Apple II + | Polymax | Maxxi |
| Apple II + | Polymax | Poly Plus |
| Apple II + | Spectrum | Microengenho I |
| Apple II + | Spectrum | Spectrum ed |
| Apple II + | Suporte | Venus II |
| Apple II + | Sycomig | SIC I |
| Apple II + | Unitron | AP II |
| Apple II + | Victor do Brasil | Elppa II Plus |
| Apple II + | Victor do Brasil | Elppa Jr. |
| Apple IIe | Microcraft | Craft IIe |
| Apple IIe | Microdigital | TK-3000 IIe |
| Apple IIe | Spectrum | Microengenho II |
| MSX | Gradiente | Expert GPC-1 |
| MSX | Sharp | Hotbit HB-8000 |
| Sinclair Spectrum | Microdigital | TK-90X |
| Sinclair Spectrum | Timex | Timex 2000 |
| Sinclair ZX-81 | Apply | Apply 300 |
| Sinclair ZX-81 | Engebras | AS-1000 |
| Sinclair ZX-81 | Filcres | NEZ-8000 |
| Sinclair ZX-81 | Microdigital | TK-82C |
| Sinclair ZX-81 | Microdigital | TK-83 |
| Sinclair ZX-81 | Microdigital | TK-85 |
| Sinclair ZX-81 | Prologica | CP-200 |
| Sinclair ZX-81 | Ritas | Ringo R-470 |
| Sinclair ZX-81 | Timex | Timex 1000 |
| Sinclair ZX-81 | Timex | Timex 1500 |
| TRS-80 Mod. I | Dismac | D-8000 |
| TRS-80 Mod. I | Dismac | D-8001/2 |
| TRS-80 Mod. I | LNW | LNW-80 |
| TRS-80 Mod. I | Video Genie | Video Genie I |
| TRS-80 Mod.III | Digitus | DGT-100 |
| TRS-80 Mod.III | Digitus | DGT-1000 |
| TRS-80 Mod.III | Kemitron | Naja 800 |
| TRS-80 Mod.III | Prologica | CP-300 |
| TRS-80 Mod.III | Prologica | CP-500 |
| TRS-80 Mod.III | Sysdata | Sysdata III |
| TRS-80 Mod.III | Sysdata | Sysdata Jr. |
| TRS-80 Mod.IV | Multix | MX-Compacto |
| TRS-80 Mod.IV | Sysdata | Sysdata IV |
| TRS-Color | Codimex | CS-6508 |
| TRS-Color | Dynacom | MX-1600 |
| TRS-Color | LZ | Color 64 |
| TRS-Color | Microdigital | TKS-800 |
| TRS-Color | Prologica | CP-400 |

## CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

| FABRICANTE | MODELO | PAÍS | LINHA |
|---|---|---|---|
| Appletronica | Thor 2010 | Brasil | Apple II + |
| Apply | Apply 300 | Brasil | Sinclair ZX-81 |
| CCE | MC-4000 Exato | Brasil | Apple II + |
| CPA | Absolutus | Brasil | Apple II + |
| CPA | Polaris | Brasil | Apple II + |
| Codimex | CS-6508 | Brasil | TRS-Color |
| Digitus | DGT-100 | Brasil | TRS-80 Mod.III |
| Digitus | DGT-1000 | Brasil | TRS-80 Mod.III |
| Digitus | DGT-AP | Brasil | Apple II + |
| Dismac | D-8000 | Brasil | TRS-80 Mod. I |
| Dismac | D-8001/2 | Brasil | TRS-80 Mod. I |
| Dismac | D-8100 | Brasil | Apple II + |
| Dynacom | MX-1600 | Brasil | TRS-Color |
| ENIAC | ENIAC II | Brasil | Apple II + |
| Engebras | AS-1000 | Brasil | Sinclair ZX-81 |
| Filcres | NEZ-8000 | Brasil | Sinclair ZX-81 |
| Franklin | Franklin | USA | Apple II + |
| Gradiente | Expert GPC1 | Brasil | MSX |
| Houston | Houston AP | Brasil | Apple II + |
| Kemitron | Naja 800 | Brasil | TRS-80 Mod.III |
| LNW | LNW-80 | USA | TRS-80 Mod. I |
| LZ | Color 64 | Brasil | TRS-Color |
| Magnex | DM II | Brasil | Apple II + |
| Maxitronica | MX-2001 | Brasil | Apple II + |
| Maxitronica | MX-48 | Brasil | Apple II + |
| Maxitronica | MX-64 | Brasil | Apple II + |
| Maxitronica | Maxitronic I | Brasil | Apple II + |
| Microcraft | Craft II Plus | Brasil | Apple II + |
| Microcraft | Caft IIe | Brasil | Apple IIe |
| Microdigital | TK-3000 IIe | Brasil | Apple IIe |
| Microdigital | TK-82C | Brasil | Sinclair ZX-81 |
| Microdigital | TK-83 | Brasil | Sinclair ZX-81 |
| Microdigital | TK-85 | Brasil | Sinclair ZX-81 |
| Microdigital | TK-90X | Brasil | Sinclair Spectrum |
| Microdigital | TKS-800 | Brasil | TRS-Color |
| Milmar | Apple II Plus | Brasil | Apple II + |
| Milmar | Apple Master | Brasil | Apple II + |
| Milmar | Apple Senior | Brasil | Apple II + |
| Multix | MX-Compacto | Brasil | TRS-80 Mod.IV |
| Omega | MC-400 | Brasil | Apple II + |
| Polymax | Maxxi | Brasil | Apple II + |
| Polymax | Poly Plus | Brasil | Apple II + |
| Prologica | CP-200 | Brasil | Sinclair ZX-81 |
| Prologica | CP-300 | Brasil | TRS-80 Mod.III |
| Prologica | CP-400 | Brasil | TRS-Color |
| Prologica | CP-500 | Brasil | TRS-80 Mod.III |
| Ritas | Ringo R-470 | Brasil | Sinclair ZX-81 |
| Sharp | Hotbit HB-8000 | Brasil | MSX |
| Spectrum | Microengenho I | Brasil | Apple II + |
| Spectrum | Microengenho II | Brasil | Apple IIe |
| Spectrum | Spectrum ed | Brasil | Apple II + |
| Suporte | Venus II | Brasil | Apple II + |
| Sycomig | SIC I | Brasil | Apple II + |
| Sysdata | Sysdata III | Brasil | TRS-80 Mod.III |
| Sysdata | Sysdata IV | Brasil | TRS-80 Mod.IV |
| Sysdata | Sysdata Jr. | Brasil | TRS-80 Mod.III |
| Timex | Timex 1000 | USA | Sinclair ZX-81 |
| Timex | Timex 1500 | USA | Sinclair ZX-81 |
| Timex | Timex 2000 | USA | Sinclair Spectrum |
| Unitron | AP II | Brasil | Apple II + |
| Victor do Brasil | Elppa II Plus | Brasil | Apple II + |
| Victor do Brasil | Elppa Jr. | Brasil | Apple II + |
| Video Genie | Video Genie I | USA | TRS-80 Mod. I |

## UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

**INPUT** foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

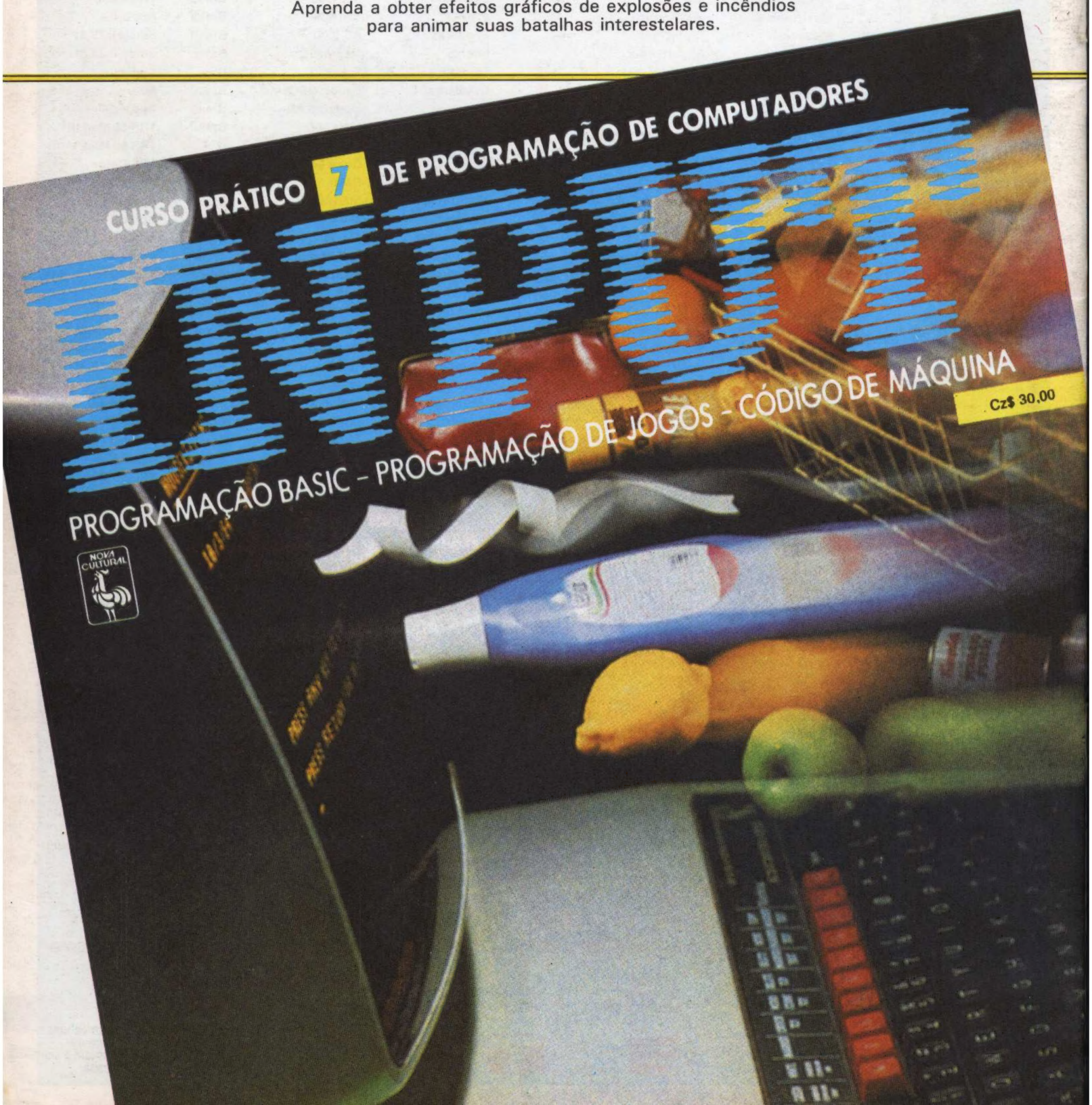
# NO PRÓXIMO NÚMERO

## PROGRAMAÇÃO BASIC

Declarações **READ** e **DATA**: o que são e como funcionam. Utilize DATA para fazer uma lista telefônica.

## APLICAÇÕES

Coloque suas despesas em ordem, empregando um programa simples e compreensível. Armazene suas finanças em fita.

## PROGRAMAÇÃO DE JOGOS

Aprenda a obter efeitos gráficos de explosões e incêndios para animar suas batalhas interestelares.